“十二五”普通高等教育本科国家级规划教材
国家精品课程教材·国家电工电子教学基地教学成果
新工科建设教学成果

微型计算机原理及应用

（第4版）

吴 宁 马旭东 张 颖 周 芳 编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

電子工業出版社

VISIT…

内 容 简 介

本书是“十二五”普通高等教育本科国家级规划教材和国家精品课程建设成果，力求做到“基础性、系统性、实用性和先进性”的统一。全书共 8 章，包括计算机基础、80x86/Pentium 微处理器、80x86/Pentium 指令系统、汇编语言程序设计、半导体存储器、输入/输出和中断、微型机接口技术和微型计算机系统的发展等。本书为任课老师提供电子课件和附录列表。

本书适合作为高校工科各专业微机原理及应用（或微机原理与接口技术）课程教材，也可作为考研参考书和从业人员的参考手册。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目(CIP)数据

微型计算机原理及应用 / 吴宁等编著. —4 版. —北京：电子工业出版社，2019.11
ISBN 978-7-121-36298-9
Ⅰ. ① 微… Ⅱ. ① 吴… Ⅲ. ① 微型计算机－高等学校－教材 Ⅳ. ① TP36
中国版本图书馆 CIP 数据核字（2019）第 068312 号

责任编辑：章海涛
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1092　1/16　　印张：23　　字数：585 千字
版　　次：2000 年 7 月第 1 版
　　2019 年 11 月第 4 版
印　　次：2019 年 11 月第 1 次印刷
定　　价：58.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：192910558（QQ 群）。

電子工業出版社

前　言

计算机技术与集成电路工艺的迅猛发展，推动了以微处理器性能为表征的新器件、新技术和新方法的不断变革，信息技术产业与新兴电子产品对社会和人类文明的影响越来越大。学习与掌握微型计算机的基本知识与应用技能，已成为人类现代文明的重要组成部分。为了与计算机技术的发展和教学改革的形势相适应，我们对教材进行了修订。

本教材自第 1 版出版以来，分别被评为**普通高等教育“十一五”国家级规划教材**、**“十二五”普通高等教育本科国家级规划教材**，受到了业界同行的广泛关注，为国内多所大学选用。对修订版的编写，我们仍然秉承一贯的宗旨，既保持多年形成的比较成熟的课程体系，适时地引入近年来微型计算机的新技术发展，同时结合国家和省精品课程建设成果及多年的教学实践，吸取国内同行师生使用前期版本的反馈意见，并参考一些国外优秀教材，力求**做到“基础性、系统性、实用性和先进性”的统一**。

微型计算机原理及其应用课程是信息类学科相关专业的重点主干课程，是后继课程学习的桥梁和纽带，是从微型计算机的基本原理与软硬件技术出发设计各类应用系统的基础，也**是后 PC 时代学习、开发和应用 ARM、DSP 及“嵌入式系统”技术的基础**。在微处理器与计算机技术飞速发展和升级换代的进程中，计算机本身的体系结构、基本工作原理并没有多少改变。基于此，本书从典型计算机体系结构、CISC 与 RISC 处理器特征入手，以 Intel 80x86/Pentium 处理器为切入点，重点讲述 8086/8088 CPU 的构成、寄存器特点、存储器管理方式以及实模式编程技术；在此基础上，介绍 Pentium 系列各处理器的发展与特点，结合扩展的指令系统给出了 Intel 架构 32 位 CPU 的编程特点，以及多媒体和保护模式编程的基本方法；同时，对 CPU 常用外围器件，如半导体存储器、典型可编程接口芯片及其相关的中断技术、接口设计方法和典型控制程序等给予详尽的介绍。**内容组织上遵循“由易及难、循序渐进、宽编窄用”的原则**，叙述上力求做到由浅入深、通俗易懂。

全书共 8 章，第 1～4 章分别介绍微型计算机系统组成、微型计算机的 CPU、汇编语言及其程序设计等基本知识；第 5～8 章介绍微机存储系统、数据传送方式、中断技术、接口芯片的相关知识与实用技术，以及典型的微型计算机系统和各种计算体系等。

本书由南京航空航天大学和东南大学优秀教学团队联合编写。其中，第 1、2 章由张颖编写，第 3、4 章由吴宁编写，第 5、7 章由马旭东编写，第 6、8 章由周芳编写。全书教学参考**学时数为 60～80**，使用时可根据具体情况选择适当的内容。本书为任课老师提供**电子课件**及由于篇幅所限不能在本书提供的**附录列表**，需要者请登录**华信教育资源网 http://www.hxedu.com.cn**，注册后免费下载。

前期版本《80x86/Pentium 微型计算机原理及应用》自出版以来，重印近 20 次，**为国内多所重点大学选为本科生教学用书**，并被列为**研究生考试的主要参考书**。借此新版出版之际，对业界同仁的信任与鼓励表示衷心的感谢。由于笔者水平有限，书中难免有错误和不妥之处，请读者批评指正。

作　者

電子工業出版社

目　录

第 1 章　计算机基础 …… 1

1.1　计算机及系统组成 …… 1

1.1.1　微型计算机硬件系统组成 …… 2

1.1.2　微型计算机软件系统 …… 7

1.1.3　微型计算机中指令执行的基本过程 …… 7

1.1.4　微型计算机性能的评估指标 …… 10

1.2　计算机中数值数据信息的表示 …… 12

1.2.1　机器数和真值 …… 12

1.2.2　数的表示方法——原码、反码和补码 …… 13

1.2.3　补码的运算 …… 16

1.2.4　定点数与浮点数 …… 17

1.2.5　BCD 码及其十进制调整 …… 20

1.3　计算机中非数值数据的信息表示 …… 22

1.3.1　西文信息的表示 …… 22

1.3.2　中文信息的表示 …… 23

习题 1 …… 24

第 2 章　微处理器 …… 25

2.1　微处理器概述 …… 25

2.2　80x86/Pentium 微处理器的内部结构 …… 28

2.2.1　8086/8088 CPU 基本结构 …… 28

2.2.2　80386 CPU 内部结构 …… 34

2.2.3　80x87 数学协处理器 …… 48

2.2.4　Pentium CPU 内部结构 …… 51

2.2.5　Pentium 系列其他微处理器 …… 56

2.3　微处理器的主要引脚及功能 …… 56

2.3.1　8086/8088 CPU 引脚功能 …… 56

2.3.2　80386 CPU 主要引脚功能 …… 61

2.3.3　Pentium CPU 主要引脚功能 …… 62

2.4　系统总线与典型时序 …… 64

2.4.1　CPU 系统总线及其操作 …… 64

2.4.2　基本总线操作时序 …… 65

2.4.3　特殊总线操作时序 …… 67

2.5　典型 CPU 应用系统 …… 69

電子工業出版社

2.5.1 8086/8088 支持芯片……69
2.5.2 8086/8088 单 CPU（最小模式）系统……73
2.5.3 8086/8088 多 CPU（最大模式）系统……74
2.6 CPU 的工作模式……77
2.6.1 实地址模式……77
2.6.2 保护模式……77
2.6.3 虚拟 8086 模式……78
2.6.4 系统管理模式……78
习题 2……78
第 3 章 微处理器指令系统……81
3.1 指令格式……81
3.2 寻址方式……83
3.2.1 寻址方式与有效地址 EA 的概念……83
3.2.2 80x86/Pentium 各种寻址方式……83
3.2.3 80x86/Pentium 存储器寻址的段约定……86
3.2.4 几种处理器寻址方式比较……87
3.3 8086/8088 CPU 指令系统……88
3.3.1 数据传送类指令……88
3.3.2 算术运算类指令……92
3.3.3 逻辑运算与移位指令……98
3.3.4 串操作指令……101
3.3.5 控制转移类指令……104
3.3.6 处理器控制类指令……111
3.4 80x86/Pentium CPU 指令系统……112
3.4.1 80286 CPU 的增强与增加指令……113
3.4.2 80386 CPU 的增强与增加指令……115
3.4.3 80486 CPU 增加的指令……117
3.4.4 Pentium 系列 CPU 增加的指令……117
3.5 80x87 浮点运算指令……120
3.5.1 80x87 的数据类型与格式……120
3.5.2 浮点寄存器……121
3.5.3 80x87 指令简介……121
习题 3……122
第 4 章 汇编语言程序设计……127
4.1 程序设计语言概述……127
4.2 汇编语言的程序结构与语句格式……129
4.2.1 汇编语言源程序的框架结构……129
4.2.2 汇编语言的语句……130
4.3 汇编语言的伪指令……134
4.3.1 基本伪指令语句……134

電子工業出版社

4.3.2 80x86/Pentium CPU 扩展伪指令 …… 146
4.4 汇编语言程序设计方法 …… 149
4.4.1 程序设计的基本过程 …… 149
4.4.2 顺序结构程序设计 …… 150
4.4.3 分支结构程序设计 …… 151
4.4.4 循环结构程序设计 …… 155
4.4.5 子程序设计与调用技术 …… 158
4.5 模块化程序设计技术 …… 167
4.5.1 模块化程序设计的特点与规范 …… 167
4.5.2 程序中模块间的关系 …… 168
4.5.3 模块化程序设计举例 …… 168
4.6 综合应用程序设计举例 …… 170
4.6.1 16 位实模式程序设计 …… 170
4.6.2 基于 32 位指令的实模式程序设计 …… 174
4.6.3 基于多媒体指令的实模式程序设计 …… 175
4.6.4 保护模式程序设计 …… 177
4.6.5 浮点指令程序设计 …… 180
4.7 汇编语言与 C/C++语言混合编程 …… 181
4.7.1 内嵌模块方法 …… 181
4.7.2 多模块混合编程 …… 181
习题 4 …… 184
第 5 章 半导体存储器 …… 188
5.1 半导体存储器概述 …… 188
5.1.1 半导体存储器的分类 …… 189
5.1.2 存储原理与地址译码 …… 190
5.1.3 主要性能指标 …… 192
5.2 随机存取存储器（RAM） …… 193
5.2.1 静态 RAM（SRAM） …… 193
5.2.2 动态 RAM（DRAM） …… 196
5.2.3 随机存取存储器 RAM 的应用 …… 198
5.3 只读存储器（ROM） …… 201
5.3.1 掩膜 ROM 和 PROM …… 201
5.3.2 EPROM（可擦除的 PROM） …… 202
5.4 存储器连接与扩充应用 …… 207
5.4.1 存储器芯片选择 …… 207
5.4.2 存储器容量扩充 …… 209
5.4.3 RAM 存储模块 …… 210
5.5 CPU 与存储器的典型连接 …… 212
5.5.1 8086/8088 CPU 的典型存储器连接 …… 212
5.5.2 80386/Pentium CPU 的典型存储器连接 …… 214
5.6 微机系统的内存结构 …… 215

電子工業出版社

5.6.1 分级存储结构 …… 216
5.6.2 高速缓存 Cache …… 216
5.6.3 虚拟存储器与段页结构 …… 218
习题 5 …… 219

第 6 章 输入/输出和中断 …… 220

6.1 输入/输出及接口 …… 220
6.1.1 I/O 信息的组成 …… 220
6.1.2 I/O 接口概述 …… 220
6.1.3 I/O 端口的编址 …… 221
6.1.4 简单的 I/O 接口 …… 224
6.2 输入/输出的传送方式 …… 225
6.2.1 程序控制的输入/输出 …… 225
6.2.2 中断控制的输入/输出 …… 228
6.2.3 直接数据通道传送 …… 229
6.3 中断技术 …… 230
6.3.1 中断的基本概念 …… 230
6.3.2 中断优先权 …… 232
6.4 80x86/Pentium 中断系统 …… 234
6.4.1 中断结构 …… 234
6.4.2 中断向量表 …… 236
6.4.3 中断响应过程 …… 237
6.4.4 80386/80486/Pentium CPU 中断系统 …… 239
6.5 8259A 可编程中断控制器 …… 242
6.5.1 8259A 芯片的内部结构与引脚 …… 243
6.5.2 8259A 芯片的工作过程及工作方式 …… 244
6.5.3 8259A 命令字 …… 247
6.5.4 8259A 芯片应用举例 …… 252
6.6 中断程序设计 …… 256
6.6.1 设计方法 …… 256
6.6.2 中断程序设计举例 …… 258
习题 6 …… 261

第 7 章 微型机接口技术 …… 265

7.1 接口技术概述 …… 265
7.2 可编程定时/计数器 …… 266
7.2.1 可编程定时/计数器 8253 …… 267
7.2.2 可编程定时/计数器 8254 …… 273
7.3 可编程并行接口 …… 274
7.3.1 可编程并行接口芯片 8255A …… 274
7.3.2 并行打印机接口应用 …… 281
7.3.3 键盘和显示器接口 …… 285

電子工業出版社

7.4 串行接口与串行通信 …… 289
7.4.1 串行通信的基本概念 …… 289
7.4.2 可编程串行通信接口 8251A …… 295
7.4.3 可编程异步通信接口 INS8250 …… 302
7.4.4 通用串行总线 USB …… 302
7.4.5 I2C 与 SPI 串行总线 …… 305
7.5 DMA 控制器接口 …… 307
7.5.1 8237A 芯片的基本功能和引脚特性 …… 307
7.5.2 8237A 芯片内部寄存器与编程 …… 309
7.5.3 8237A 应用与编程 …… 312
7.6 模拟量输入/输出接口 …… 314
7.6.1 概述 …… 314
7.6.2 并行和串行 D/A 转换器 …… 315
7.6.3 并行和串行 A/D 转换器 …… 321
习题 7 …… 329
第 8 章 微型计算机系统的发展 …… 332
8.1 微型计算机体系结构及系统总线 …… 332
8.1.1 微型计算机体系结构 …… 332
8.1.2 系统外部总线 …… 335
8.2 工作站 …… 338
8.2.1 配置和功能 …… 338
8.2.2 分类 …… 338
8.2.3 工作站的特点 …… 339
8.3 服务器 …… 340
8.3.1 分类 …… 340
8.3.2 硬件特点 …… 341
8.3.3 外形 …… 342
8.3.4 计算机、工作站和服务器 …… 343
8.4 SoC 与嵌入式系统 …… 343
8.4.1 SoC …… 343
8.4.2 嵌入式系统 …… 345
8.5 多核处理器 …… 347
8.5.1 发展历程 …… 347
8.5.2 多核技术 …… 348
8.5.3 多核处理器开发应用 …… 349
8.6 并行计算与分布式计算 …… 349
8.6.1 并行计算 …… 349
8.6.2 分布式计算 …… 351
8.6.3 云计算、集群计算及网格计算 …… 353
参考文献 …… 356

電子工業出版社

第 1 章　计算机基础

计算机是人类发明的最引人瞩目的设备，影响了整个世界，其中，微型计算机是性能提高最快、应用最广泛的数字计算机。通用微型计算机配有系统软件和外围设备，一般可用于数值计算、信息处理和智能控制等诸多方面。本章主要介绍计算机系统的基本构成、计算机系统的性能评价指标以及计算机数据处理的基础。

1.1　计算机及系统组成

计算机的结构大体有两类：一是哈佛（Harvard）结构，二是冯·诺依曼（Von Neumann）结构（也称为普林斯顿结构）。

哈佛结构是由哈佛大学的物理学家 A.Howard 提出的，世界上第一台通用计算机 ENIAC，就采用了哈佛结构。哈佛结构是程序和数据空间独立的体系结构，不仅采用不同的存储模块，还可以通过各自的总线进行访问。其优点是可以并行访问程序和数据，从而提升了运行效率；其缺点是硬件结构复杂，对输入/输出设备的连接与处理要求高。

冯·诺依曼结构是由美籍匈牙利数学家冯·诺依曼提出的。相对于哈佛结构，其程序和数据共享存储空间和总线，由此带来的缺点是，将分时实现程序和数据的访问，影响了数据处理的速度；但其显著的优点是结构简单、易于实现。

冯·诺依曼结构的基本思想如下：

① 计算机应由运算器、控制器、存储器、输入设备和输出设备五大部分组成。

② 存储器不但能存放数据，也能存放指令。计算机具有区分数据和指令的能力，而且数据和指令均以二进制数形式存放。

③ 将事先编好的程序存入存储器中，在指令计数器控制下，可以实现程序的自动运行（执行程序）。

由于体系结构实现的便利和总线结构的易于扩展，使得冯·诺依曼结构成为 PC 机（Intel 兼容机）的主流体系标准。本书的主要内容也是针对冯·诺依曼体系结构的微型计算机。

计算机系统包括读取并执行程序的中央处理单元 CPU，保存程序和数据的存储器，以及连接外围设备的各类转接模块。这些转接模块（如磁盘控制器、图形适配器等）会使 CPU 与显示器、打印机、Internet 等外部设备之间的通信变得更容易，计算机系统结构如图 1-1 所示。

CPU 通常为单个芯片，又称为微处理器，可依据其使用的指令系统，分为复杂指令集 CISC（Complex Instruction Set Computing）和精简指令集 RISC（Reduced Instruction Set Computing）处理器。CISC 是早期的微处理器（x86 系列及其兼容 CPU）使用的指令集。RISC 是在 CISC 指令系统基础上发展起来的，比 CISC 的指令种类少，可采用超标量和超流水线结构，大大增加了并行处理能力，在同等频率下，采用 RISC 架构的 CPU 比 CISC 架构的 CPU 性能高很多。典型的 RISC 处理器有 ARM 和 MIPS 处理器。为了与同系列低端微处理器保持兼容，大多数个人计算机中使用的微处理器仍使用 CISC 指令集，但考虑到 RISC 在设计和指

電子工業出版社

令结构等方面的优势，现在的 CISC 结构的微处理器都结合了部分 RISC 技术，形成以 RISC 为内核、CISC 为外围的微处理器，如典型的有 Intel Pentium 系列和 AMD K6 系列等。

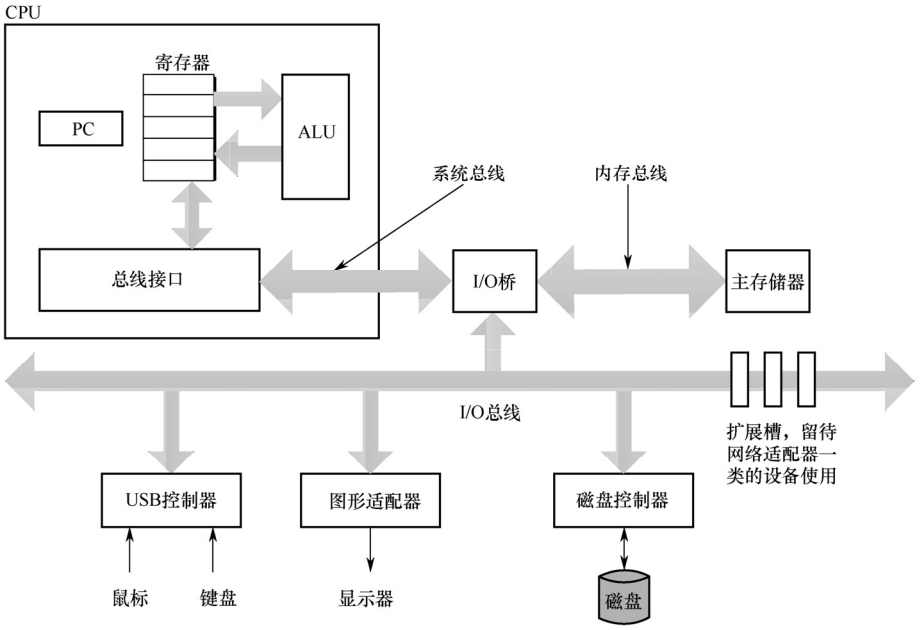


图 1-1　计算机系统结构

RISC 和 CISC 处理器的主要差异在于以下几点。

① RISC 指令结构固定简单，利于流水线技术，但实现特殊功能时效率可能较低；CISC 的指令系统比较丰富，有专用指令来完成特定的功能，因而处理特殊任务效率较高。

② RISC 对存储器操作有限制以利于控制的简单化；CISC 的存储器操作指令形式多，操作直接。

③ RISC 指令数量少和寻址方式的简单使得指令的编码紧凑、译码速度快、控制逻辑简单，因而硬件译码电路相对简单；CISC 的微程序控制指令条数多、形式复杂，硬件译码难以实现。

④ RISC 汇编语言程序一般需要较大的内存空间，实现特殊功能时程序复杂，不易设计；CISC 汇编语言程序编程相对简单，科学计算及复杂操作的程序设计相对容易，效率较高。

⑤ RISC 微处理器结构简单，布局紧凑，因而面积小、功耗低，设计周期短，且易于采用最新技术；CISC 微处理器包含丰富的电路单元，因而功能强、结构复杂、面积大、功耗相对较大，设计周期长。

1.1.1　微型计算机硬件系统组成

通用微型计算机的硬件系统由 5 部分组成：① 微处理器（CPU）；② 内存储器（RAM、ROM）；③ 总线（地址总线 AB、数据总线 DB、控制总线 CB）；④ 接口芯片（I/O 接口）；⑤ 输入/输出设备（外设，I/O）。图 1-2 表示典型的通用微型计算机硬件系统结构。

電子工業出版社

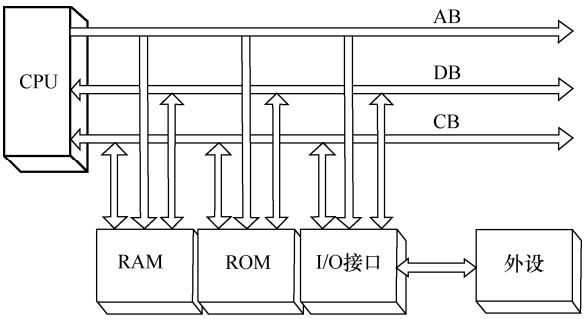


图 1-2　通用微型计算机硬件系统结构

1．微处理器 CPU

微处理器是整个微型计算机硬件的控制指挥中心。不同型号的微型计算机性能的差别，首先体现在微处理器性能的不同。微处理器性能又与它的内部结构、硬件配置有关。但无论是哪种微处理器，基本部件是相同的，可分为运算部分和控制部分。运算部分由算术逻辑单元 ALU（Arithmetic Logic Unit）、状态寄存器 FR（Flags Register）和通用寄存器组 RS（Register Series）组成；控制部分则包括程序计数器 PC（Program Counter）、指令寄存器 IR（Instructional Register）、指令译码器 ID 以及控制逻辑 PLA 等。图 1-3 是通用微处理器基本组成示意。

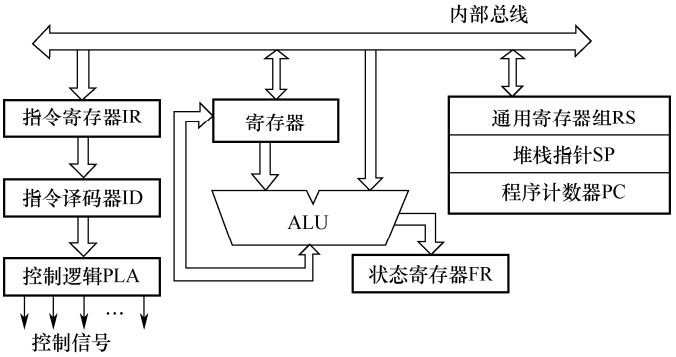


图 1-3　通用微处理器组成示意

① ALU 是微型计算机运算部分的核心。在控制信号作用下，ALU 可完成加、减、乘、除四则运算，还可进行与、或、非和异或等逻辑运算。ALU 只能用于实现整数的运算，而浮点数的运算通常需要专用的浮点运算器来实现。

② RS 用来加快运算和处理速度。访问寄存器要比访问存储器快得多，因此在需要重复使用某些数据或中间结果时，可将其暂时存放在寄存器中，避免反复访问存储器，以提高执行速度，其中的累加器 ACC 因为通常作为 ALU 的运算操作数，因而具有更快的执行速度。Intel 的 X86 系列的寄存器具有专用寄存器的特性，RISC 处理器的寄存器则是真正的通用寄存器。

③ FR 记录计算机运行的运算器和控制器的状态值，可根据这些状态值控制 CPU 的运行，是决定程序分支的重要参考量。

④ 堆栈和堆栈指针 SP。堆栈是一组寄存器或指定存储器中的某一区域。在堆栈中信息的存入（进栈 Push）和取出（弹出 Pop）过程好像货物堆放的过程，最先存放的货物堆在底部，而取货时只能将最后存放的先取出；最后存放的货物，堆放在顶部，因而最先取出。这

電子工業出版社

种方式称为“后进先出”（LIFO）或“先进后出”（FILO）。在计算机中广泛使用“堆栈”作为信息的一种存取方式，尤其是在子程序（过程）调用和中断处理中经常使用。

堆栈指针 SP（Stack Point）用来指示栈顶，其初值由程序员设定。在堆栈操作中，SP 值会自动变化，其变化方向依据堆栈的类型而定。堆栈可分为向上生长型和向下生长型。对于向下生长型堆栈，将数据压入堆栈时，地址自动递减；当数据从栈中弹出时，地址则递增。对于向上生长型堆栈，数据压入和弹出堆栈时，地址则分别是递增和递减。

⑤ PC 指示当前要执行的指令地址码。程序中的各条指令一般是按执行的顺序存放在存储器中。开始时，PC 中的地址码为该程序第一条指令所在的地址编号。每取出一条指令，PC 将自动加上该指令所占字节长度以指向第二条指令所在地址，这样就能顺序执行程序中的各条指令。在需要改变执行顺序时，PC 根据程序的安排，设置新的值以指向新的指令地址。因此，PC 是维持微处理器有序地执行程序的关键寄存器，是任何微处理器不可缺少的。

⑥ IR、ID 和控制逻辑 PLA。这部分是整个微处理器的指挥控制中心，对协调整个微型计算机有序工作极其重要。它根据用户预先编好的程序，依次从存储器中取出各条指令，放在指令寄存器中，通过指令译码（分析）确定应该进行什么操作，然后通过控制逻辑在确定的时间，向确定的模块发出控制信号。这部分对微处理器的设计者来说是核心部分，而使用微处理器的用户可以不必过多关心它。

需要指出的是，上文所述的 CPU 基本结构主要是针对通用处理器（典型的如 Intel 和 AMD 的 CPU、MIPS 处理器），目前应用广泛的还有微控制器 MCU（典型的如 MCS-51 单片机）、数字信号处理器 DSP（典型的如 TMS320VC5000）和嵌入式微处理器（典型的如 ARM9 或 ARM Cortex A-8），它们在各自的系统中承担主控处理器的作用，核心部分和通用处理器类似，但结构功能略有差异。

2．内存储器

微计算机系统的内存储器由大规模（LSI）或超大规模（VLSI）集成电路芯片构成，主要用来存储数据和程序。具体来讲，内存储器中存放着两类信息，一类是待处理的数据和运算结果，另一类是处理数据的程序。

内存储器的工作过程大致如下：在执行程序前，预先把程序和原始数据存放于内存储器中，在处理过程中，由它向控制器提供指令代码，然后根据处理需要，随时向运算器提供数据，并把运算结果或中间结果存储起来，从而保证机器能按照程序自动地进行工作。因此，在计算机结构中，内存储器是极其重要的部件之一。

内存储器要保存数以万计的数据，如何把这些数据有规律地存放，以便存取数据时方便、迅速，是很重要的问题。其存放方法是，把内存储器分成一个个单元。每个单元存放着固定位数的二进制数据，通常以字节（8 位二进制数）为单位。每个单元都有一个编号与之对应，称为地址（地址码），只要指明地址，就可以到存储器的成千上万个单元中按所指地址存入或取出所需数据。

图 1-4 表示一个有 m 个单元（字），每个单元为 1 字节的存储器。地址从小到大顺序编号，即 0～m−1。地址码用二进制编码表示。由于二进制数读/写不方便，常用十六进制数表示地址。现代通用微型计算机的字长为 8、16、32、64 位共 4 种规格。如市场销售的计算机存储器存储容量 64 KB，即表示该存储器有 64×1024=65536 字节的容量。

对存储器某地址的数据进行读或写（即取数或存数）操作统称为访问（Access）。为了实

電子工業出版社

现存数和取数的目的，存储器除了核心部分的存储体，还需要一些辅助电路来支持其工作。这些部件包括存储器地址寄存器（MAR）和地址译码器，存储器数据寄存器（MDR）和控制整个存储器协调工作的控制电路，如图 1-5 所示。

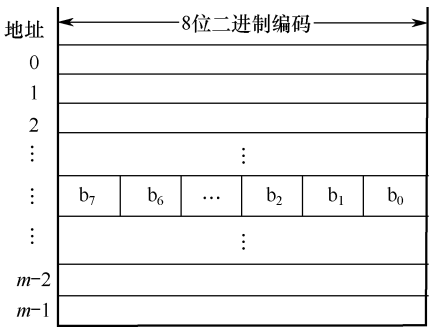


图 1-4　存储单元组织及地址表示

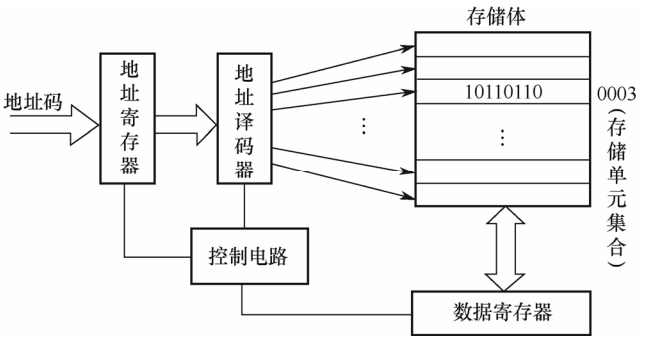


图 1-5　存储器结构

当 CPU 部件需要访问存储器的某个地址时，首先把地址码送到存储器的地址寄存器，经过译码电路选中相应地址单元。取数时，取出的数据送到存储器的数据寄存器，以供使用该数据的 CPU 适时取走。存数时，把要存入的数据送入数据寄存器，通过写操作，将其存入指定地址的单元中。控制电路则根据本次访问存储器是取或存数，产生一定顺序的控制信号和脉冲。控制电路把存储器内部各部件有机联系在一起，并使它们协调地按一定节拍工作。

设存储器 0003 号单元已存放二进制信息 10110110。现在要读出此信息的过程是：先将地址 0003 送往地址寄存器，地址译码器将地址译码后产生选中 0003 号单元的信号，随后在控制电路作用下发出读控制信号，0003 号单元的内容 10110110 被读出并送往数据寄存器。若是写数据，则过程类似，只是当译码器产生选中地址单元信号后，控制电路将发出写控制脉冲，将数据寄存器内容写入相应单元中。注意，存储单元的地址及单元的内容是不相同的。

由于每个地址单元仅存储 1 字节，多字节数将被存储为连续的字节序列，其地址为所使用字节中的最小地址。这里会涉及多字节存放顺序的问题，有些计算机系统会选择在内存中按照最低有效字节到最高有效字节的顺序存放（即低地址对应低字节），其他系统则采用相反的处理方式（即低地址对应高字节），前一种规则——最低有效字节在最前面的方式，称为小端法（little endian），后一种规则——最高有效字节在最前面的方式，称为大端法（big endian）。Intel 兼容机采用的都是小端模式，有些单片机采用的是大端模式，有的 ARM 系统可以通过硬件配置选择大端或小端模式。

微型计算机存储器通常有两类：随机访问存储器 RAM（Random Access Memory）和只读存储器 ROM（Read Only Memory）。RAM 可以随时写入内容，亦可进行修改。ROM 事先存储规定好的内容，只能将规定内容读出，不能改写。

3．总线

计算机的各组成部分必须有机地连接在一起，才能相互协调地工作。连接的方法有多种，最直观和最简单的方法是把任何两个有联系的部件用导线直接连接起来。这种连接虽然有直接传输、独立使用和传输速度快等优点，但其致命弱点是，各组成部分相互关系繁多，致使连线多且乱。因此，微处理器及微型计算机系统一般采用“总线”（Bus）结构把各组成部分连接起来。

電子工業出版社

总线，实际上是一组导线，是各种信息线的集合，用来作为各组成部分共同使用的“公路”。微型计算机中的总线一般指数据总线 DB（Data Bus）、地址总线 AB（Address Bus）和控制总线 CB（Control Bus），见图 1-2。

数据总线用来传输数据，通常包括 CPU 与内存储器或输入/输出设备之间、内存储器与输入/输出设备或外存储器之间交换数据的双向传输线路。地址总线用来传输地址，一般是从 CPU 传输至内存储器、输入/输出设备。控制总线用来传输控制信号、时序信号和状态信息等，其中每根线的方向是一定的、单向的，但作为整体，控制总线是双向的。

微型计算机系统的总线可按连接对象的不同层次分为以下 3 类。

① 内总线：又称为板内总线，它把 CPU、ROM、RAM、基本 I/O 接口、定时器及总线控制逻辑等部件连成一个系统，成为微机的基本硬件系统。内总线是微机系统的一级总线。

② 系统总线：又称为板间总线，它由 CPU 通过总线控制逻辑进行管理，在该总线上装有 5～8 个通用 I/O 扩展插槽，不同设备的接口电路都将通过该扩展槽与主机连接。系统总线是微机插板级总线，如 8 位 PC/XT 总线，16 位 ISA 总线，32 位 VESA 总线和 32 位或 64 位 PCI 和 PCIe 总线等。

③ 外部总线：用于设备与设备之间连接，常用的有 RS-232、IEEE-488、IEEE1394 和 USB 总线等。

前面所述的三总线结构也称为单总线结构，早期的大多数微机和 Intel 兼容机都采用这种结构（冯・诺依曼结构），其逻辑简单，成本较低。另一方面，RISC 处理器中大量采用了双总线（哈佛结构），如图 1-6(a)所示。双总线结构中，I/O 和存储器各自拥有连接至 CPU 的总线通路，相当于拓展了总线带宽，提高了总线的数据传输速率，配合指令集的设计，可提升数字信号的处理速率。当然，在这样的结构下，CPU 要负责两条总线的通信，往往需要增加专门的 I/O 管理芯片来分担 CPU 的管理任务。

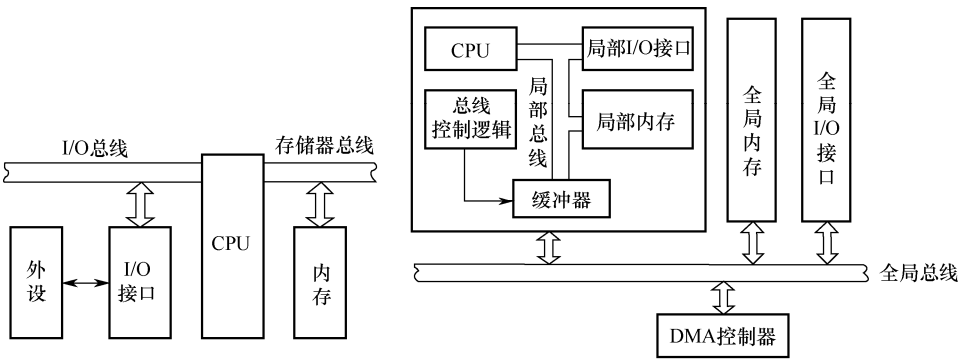


图 1-6　微型计算机的三种总线结构

现代多数的微机和工作站采用的总线结构称为双层或多层总线，如图 1-6(b)所示。双层（多层）总线的设计思路是，速度差异较大的器件使用不同总线，而速度相近的器件使用同一类总线，这种结构不仅可以增加总线挂载数量，在系统规模增加时，还可以使总线设计相对简单，能充分发挥总线的效率。

例如，图 1-6(b)中的双层总线分为局部总线和全局总线，前者连接 CPU 和局部 I/O 接口及局部存储器（速度较高），后者连接全局 I/O 接口、全局存储器及局部总线部分（速度相对低）。当 CPU 通过局部总线进行通信时，其工作方式与单总线相同。当 CPU 需要访问全局 I/O

電子工業出版社

或全局存储器时，必须由总线控制逻辑统一进行安排，此时 CPU 就是系统的主控设备。这种总线结构可以实现双层总线（局部总线和全局总线）并行工作，同时增加了等效总线带宽，提高了系统数据处理和数据传输的效率。

4．输入/输出设备与接口芯片

输入/输出设备是微型计算机系统与周围世界通信联系的渠道。微型计算机的输入/输出设备种类繁多。输入设备有键盘、条形码识别装置、手写或触屏设备以及模/数转换器等。输出设备有显示器、打印机、音频输出设备以及数/模转换器等。常用作外存储器的磁盘、移动存储设备既是输入又是输出设备。输入/输出设备又称外围设备（Peripheral Equipment）。

尽管输入/输出设备种类繁多，但具有两个共同特点：① 常采用机械或电磁原理工作，所以速度较慢，难以与纯电子的 CPU 及内存储器的工作速度相匹配；② 要求的工作电平常常与 CPU 和存储器等采用的电平不一致。为了把输入/输出设备与计算机的 CPU 连接起来，需要一个中间环节——接口（Interface），用来锁存、变换、隔离和外设选址，以保证信息和数据在外设与 CPU 和内存之间正常传送，这类功能电路通常集成为可编程接口芯片。

1.1.2 微型计算机软件系统

仅仅具备上述硬件模块的计算机系统是无法供用户使用的。为了运行、管理和维护计算机所需的程序总和就是计算机软件。计算机软件可分为系统软件和应用软件两大类。系统软件是用来支持应用软件的开发与运行的，包括操作系统、实用程序和各种语言处理程序。应用软件是用来为用户解决某种应用问题的程序及有关的文件资料。

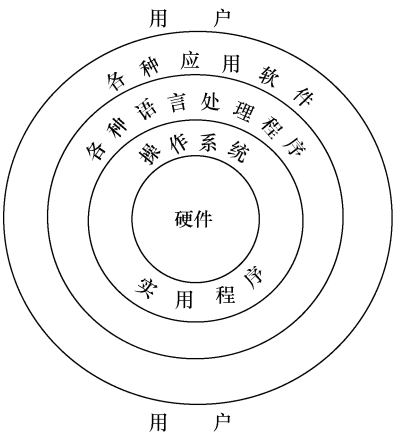


图 1-7　微型计算机软件系统的层次关系

硬件、系统软件和应用软件相互之间的关系如图 1-7 所示，表明计算机的基础是硬件，在此基础上建造了一层系统软件（操作系统和实用程序），再在系统软件基础上又建造各种语言处理程序。最外层是用户，用户通过键盘、显示器等交互设备使用应用软件，应用软件通过系统软件访问底层硬件，最终实现用户所需功能。

1.1.3 微型计算机中指令执行的基本过程

1．指令与程序简介

指令是人们规定计算机执行特定操作（加、减、乘、除、移位等）的命令。微处理器就是根据指令指挥和控制计算机各部分协调地动作，以完成指令所规定的操作。计算机全部指令的集合称为计算机指令系统。指令系统定义了计算机的处理能力，不同型号的计算机有不同的指令系统，从而形成各种型号计算机的独自特点和差异。

为解决某一具体问题，将指令和数据编写成一个相互联系的序列（在高级语言中是由语句和数据组成的）称为程序。如果所用指令编写的程序是计算机能直接理解和执行的二进制代码形式，称为机器语言，相应的程序称为机器语言程序。机器语言对使用者来说烦琐且容

電子工業出版社

易出错，为克服这些缺点，就用一组便于记忆、缩写、简写英文字母构成的符号（助记符）来代替机器语言指令，这就是汇编语言。汇编语言便于人们记忆和交流，但计算机仍不能直接识别，为此在交付计算机执行前，必须将它翻译成机器语言的目标程序，这个过程称为汇编。关于用汇编语言编制程序的具体内容，将在第 4 章详细介绍。

2．指令的执行

微型计算机执行一条指令所需的时间称为指令周期。指令周期分成两个阶段：取指令阶段和执行指令阶段。其主要任务如下：

① 取指令阶段。根据 PC 中的值从存储器读出指令并送到 CPU 的指令寄存器 IR，PC 则自动修改，指向下一条指令地址。

② 执行指令阶段。将 IR 中的指令操作码译码，并根据译码产生相应的控制电位和定时节拍信号以执行指令所规定的操作。

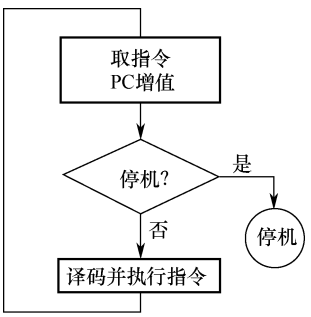


图 1-8　程序中指令执行过程

程序中指令的执行过程就是上面两个阶段的交替过程，直至遇到停机指令时才使整个机器停止运行，如图 1-8 所示。

由于程序和数据存放于内存储器里，所以在执行指令时，CPU 必须频繁地访问存储器。在图 1-9 所示的指令执行示意图中，给出了 CPU 访问内存储器的相关寄存器的情况。地址寄存器 MAR 存放着将访问的地址信息，数据寄存器 MDR 存放从内存储器读出的或拟向内存储器写入的数据，指令寄存器 IR 保存待处理指令的操作码。

下面以执行一条“MOV　A, 00H”指令为例，说明该指令在该假想模型机上的执行过程。该指令机器码为（3EH, 00H），即 2 字节，并存放在内存储器的第 100H、101H 单元中，其第一字节为操作码 3E，表示该指令是立即数传送指令，后面一字节是立即数。其指令功能是将立即数 00H 送往累加器 ACC。根据图 1-9 所示，该指令的第一字节存于内存储器的第 100H 号单元，因此在执行本指令前 PC 值应为 100H。

为了讨论方便，这里使用下列符号：① (x)、[x]表示 x 单元存储的内容，x 可以是寄存器名称，如 ACC、PC 等，也可以是存储单元地址，如[300]表示存储器第 300 号单元内的数据（内容）。② →表示数据传送流向。

取指令阶段：

（1）(PC)→ MAR；　（2）PC+1→ PC；　（3）CPU 发“读”信号；

（4）[100H]→MDR；　（5）(MDR)→IR；　（6）经译码由控制逻辑发相应控制信号。

执行指令阶段：

（1）(PC) →MAR；　（2）PC+1→PC；　（3）CPU 发“读”信号；

（4）[101H]→MDR；　（5）(MDR)→ACC。

3．指令的流水线执行

早期的 CPU 在执行指令时，采用非流水线操作方式。假定一条指令需要 5 个操作步骤完成，即取指、译码、读操作数、执行、写操作数，每一个操作步骤需用一个时钟周期完成，则该条指令的执行需要 5 个时钟周期时间。

電子工業出版社

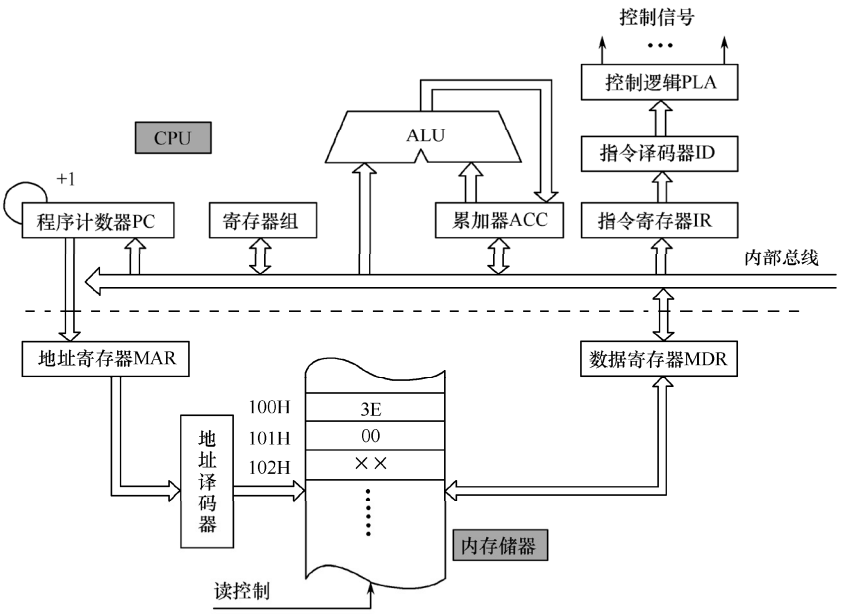


图 1-9　指令执行示意图

RISC 处理器通常采用流水线（Pipeline）技术来优化指令执行速度，图 1-10 为典型的 RISC 五级流水线结构。

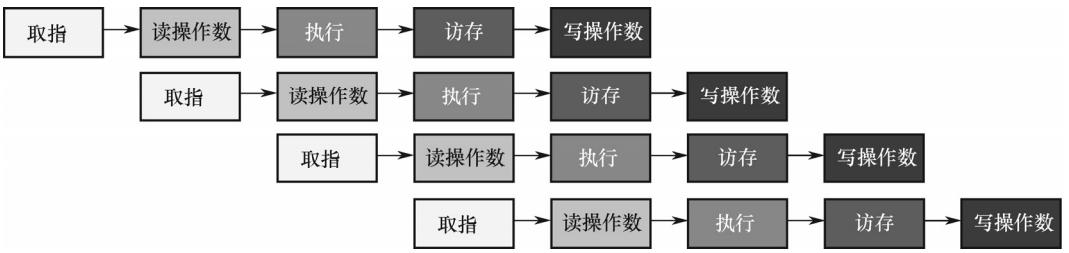


图 1-10　五阶段流水线

在这种结构中，当前指令的取指阶段一完成，下一条指令立即开始执行，流水线就是这样实现的。在不改变实现技术的情况下，这种方法可以高效地实现速度翻倍。速度的提升依靠的是更加高效地利用处理器的功能部件而不是提高时钟频率。值得指出的是，典型的 RISC 结构中译码阶段并不是必需的，因为它能与读操作数阶段合并。

流水线的优化与体系结构和指令集的设计相关，特定类型指令（如分支和其他控制程序流的指令）的处理方式在很大程度上影响了流水线的长度。取指周期的时间与取操作数、执行和保存操作数等阶段的时间之和的比率，在决定流水线的最优长度上起了至关重要的作用。

本书介绍的 Intel 系列微处理器首次在 80486 芯片中采用流水线技术。流水线由 5～6 个执行不同功能的部件组成，基于这种硬件设计，将一条指令分解为 5～6 个操作步骤，再将这些操作步骤按其操作的先后顺序，分配给具有不同功能的部分去并行执行，这样就能实现在一个 CPU 时钟周期内完成一条指令的操作目标。因此，流水线技术大大提高了 CPU 的运算速度，有效地利用了 CPU 的硬件资源，在时间和空间上挖掘出 CPU 的潜能。

80486 只有一条流水线，其流水线操作分成预取指令、首次译码、二次译码、执行指令和写回 5 个步骤。

電子工業出版社

Pentium CPU 中具有各自独立操作功能的流水线，即 U 流水线和 V 流水线，能够在单个时钟周期里执行两条指令。使用多条相互独立的指令流水线称为超标量流水线技术。

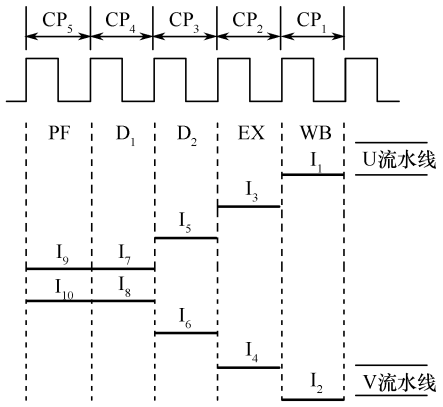


图 1-11　第 5 个时钟周期的 Pentium 流水线执行状态

Pentium CPU 的超标量流水线中的 5 个操作步骤分别是：① PF 指令预取；② D_1 指令译码（首次译码）；③ D_2 地址生成（二次译码）；④ EX 执行指令——ALU 和高速缓存（Cache）存取；⑤ WB 写回。

运行时，两条指令按照流水线的 5 个操作步骤以并行方式执行，图 1-11 给出了 Pentium CPU 中指令流从开始到经历了第 5 个时钟周期时，流水线中各条指令即时的执行状态。

并行执行两条指令的过程称为“配对”过程。U 流水线可以执行 Intel 架构中的任何指令，而 V 流水线则只能执行“指令配对规则”所规定的简单指令。在指令配对时，发送给 V 流水线的新指令总是发送给 U 流水线指令之后的下一条指令。

流水线的第 1 个操作步骤是预取操作，即从片内的指令 Cache 或内存中预取指令。流水线的第 2 个操作步骤是首次译码。在此期间，由两个并行译码器进行译码，并发出紧挨着的两条指令。译码器根据“指令配对规则”，决定是发送一条指令还是两条指令。和 80486 类似，Pentium CPU 也需要一个额外的首次译码时钟来对指令前缀进行译码，其以每个时钟发出一个前缀的速率将前缀发送到 U 流水线，且不需要配对。在所有指令前缀都发出之后，再发送基本指令，并且还要根据配对原则进行配对。

紧随 Pentium CPU 的首次译码操作步骤之后，在二次译码步骤期间，进行的是操作数在内存中地址的计算。在执行操作步骤期间，执行 ALU 的操作和数据 Cache 的存取操作，这都需要一个以上的时钟周期时间。Pentium CPU 流水线操作的最后一个操作步骤是写回操作。在此期间，指令不仅能够修改 CPU 的状态，而且还要完成指令规定的操作。

1.1.4　微型计算机性能的评估指标

一台微型计算机性能的优劣是由其系统结构、指令系统、硬件组成、外部设备、软件配备是否齐全等因素决定的。只有综合各项指标才能正确评价与衡量计算机性能的高低。下面介绍几项主要评估指标。

（1）CPU 字长

计算机的字长决定了计算机内部一次可以处理的二进制代码的位数。它决定着计算机通用寄存器、加法器、数据总线等部件的位数，因此直接影响硬件成本。一般地，一台计算机的字长由下列一个或两个条件所决定：① ALU 一次能处理的二进制编码位数；② 数据总线的宽度；③ 内存储器的字长。

字长越长，一个字能表示的数据精度越高。在完成同样精度的运算时，字长较长的计算机比字长短的计算机速度快。为了兼顾精度与硬件成本，有些计算机允许采用变字长运算。

一般情况下，CPU 内部数据总线宽度和计算机对外的数据总线宽度是一致的。早期有些 CPU 为了配合外设的总线宽度，设计内部字长和外部数据总线宽度不一致。如 Intel 8088

电子工业出版社

微处理器就是内部总线宽度为16，而外部数据总线宽度是8位的芯片，称为“准16位”芯片。而后期的CPU，则提供比内部数据宽度更高的总线宽度来访问内存，以提高访问速度。如Pentium处理器的字长为32位，但其访问内存的外部数据总线宽度为64位。

（2）内存储器与高速缓存

内存储器容量是衡量存储二进制信息量的重要指标。通常用字节表示内存储器的容量，如64 MB。内存储器容量范围大小根据其用途、成本及价格等因素选择。

内存储器的速度用存取周期来衡量。存储器执行一次完整的读/写操作所需要的时间称为存取周期，通常情况下，也就是从存储器中连续存或取两个字所用的最小时间间隔。半导体集成电路存储器的存取周期目前约为几纳秒（ns）。

内存与CPU之间的运行速率会存在差异，因此便出现了内置的高速缓存（Cache），来协调两者的速度匹配。内置的高速缓存容量越大，CPU 的性能相对也会提升很多，但是考虑到CPU芯片面积的约束，高速缓存的容量通常远小于内存，目前其容量基本为MB级。

（3）CPU指令执行时间

指令执行时间的长短反映CPU运算速度的快慢。因为执行不同的指令所需的时间不同，就产生了如何评估计算速度的问题。常用的方法有如下两种。

① 指令执行的平均速度值

根据不同类型指令在计算过程中出现的频繁程度乘上不同系数，可求得统计平均值，这里的运算速度指平均速度MIPS（Millions of Instruction Per Second），以百万条指令/秒作为单位；其全名为Drytone V1.1 MIPS，即运行一个名为Drytone的测试程序，测试性能规范到VAX1.1计算机，如486DX性能相当于每秒执行2700万条指令的VAX1.1计算机，称为27VAX MIPS。

② CPU的主频

主频是CPU的时钟频率，很大程度上决定了计算机的运算速度，是CPU的标称频率。如80486时钟频率为33～66 MHz，Pentium则为60～133 MHz，Pentium MMX CPU时钟频率为230 MHz以上。

在80486 CPU出现后，由于CPU频率不断提高，外接设备的速度却受到工艺的限制，不能承受更高的频率，因此限制了CPU频率的进一步提高，出现了倍频技术。该技术能够使CPU内部工作频率变为外部频率的倍数，从而可通过提升倍频而达到提升主频的目的。当主频升至3 GHz后，功耗和散热等因素使得频率不适宜进一步提升，为了优化处理器的工作速度，采用的解决办法是多核设计。

（4）系统总线的传输速率

系统总线的传输速率直接影响计算机输入/输出的性能，与总线的数据线宽度及总线周期有关，以MB/s为单位。早期的ISA总线传输速率仅为5 MB/s，扩充的32位总线EISA传输速率为20 MB/s，现在广泛使用的PCI局部总线传输速率为133 MB/s（32位）或266 MB/s（64位），PCIe总线的传输速率为512 MB/s、2 GB/s、4 GB/s、8 GB/s（数据通道数量为1、4、8、16）。

（5）优化的内部结构

CPU体系的发展过程中采用了许多新技术，优化了其软件、硬件结构，提升了CPU的整体性能。在这些新技术中，指令流水线结构提升了系统的并行处理能力，从而加快了指令执行时间，而高速缓存（Cache）结构加快了CPU访问数据和指令的速度，这两项技术

电子工业出版社

的应用可以说是优化 CPU 性能的关键技术。除此之外，高端 CPU 往往在其设计实现中进行了多方面、多层次的优化，显著提升了其性能。优化技术包括：超流水线和超标量技术、乱序执行技术、分支预测执行技术等。

超流水线是指某些 CPU 内部的流水线超过通常的 5～6 步，如 Pentium CPU 的流水线为 14 步，流水线设计的步数越多，其完成一条指令的速度越快。超标量是指在 CPU 中有 1 条以上的流水线，并且每个时钟周期内可以完成一条以上的指令。

乱序执行和分支预测执行技术是高端 CPU 进行动态处理的核心技术，动态处理采用多路分流预测算法，在取指同时寻找将要执行的指令，该技术可加速向处理器传送任务，通过提前判读并执行有可能需要的指令的方式提高执行速度，可使高端 CPU 的处理能力得到充分的发挥，从而提升系统性能。

（6）I/O 设备配备情况

在现代计算机系统中，I/O 设备配备的性能对整体系统的综合表现具有重大的影响。在微型计算机系统中，外存储器容量、打印机型号、显示屏幕的分辨率，都是 I/O 设备选用中要考虑的问题。

（7）软件配备情况

软件是计算机系统不可缺少的重要组成部分，配备是否齐全直接关系到计算机性能的好坏、效率的高低。这里主要指操作系统、高级语言和汇编语言的种类、应用软件等。

1.2 计算机中数值数据信息的表示

1.2.1 机器数和真值

计算机在本质上只能识别 0、1 表示的二进制数码。为了表示正数和负数，专门选择一位二进制数来表示数的符号，该位为 0，表示正号；该位为 1，表示负号，通常选择最高位作为符号位。也就是说，数的符号在计算机中也被数值化了。一个数在机器（计算机）中的表示形式称为机器数，而数本身的实际值称为真值（机器数真值）。真值可以用二进制数表示，也可用十进制数表示，但根据习惯，常用十进制数表示，如图 1-12 所示。

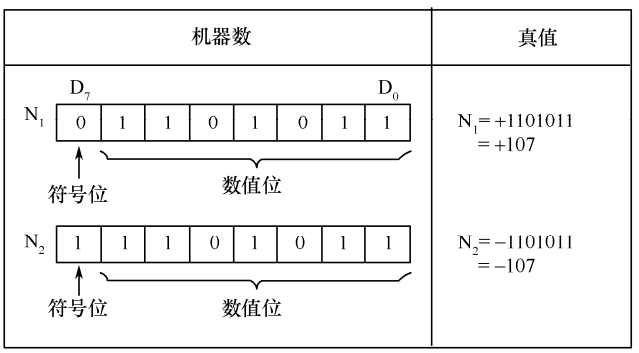


图 1-12 机器数真值

机器数有如下 3 个特点：

（1）机器数的正、负号是数值化的

（2）机器数所能表示的数的范围受到机器（计算机）字长的限制。

那么，什么是计算机字长呢？我们先来解释几个计算机中的常用术语。

① 位（bit）：计算机所能表示的最小数据单位，只有两种状态——0 和 1。

② 字节（Byte）：8 位二进制数称为 1 字节，是计算机处理数据的基本单位（B）。在计算机中，其存储器容量大小常以字节数的多少来度量。常用的 4 种度量单位是 KB、MB、GB 和 TB，其大小分别为：

1 KB=2^{10} B=1024 B　　1 MB=$2^{10}\times2^{10}$ B＝1024 KB

1 GB=$2^{10}\times2^{10}\times2^{10}$ B =1024 MB　　1 TB=$2^{10}\times2^{10}\times2^{10}\times2^{10}$ B =1024 GB

③ 字（Word）：作为单位，指 16 位二进制数，也将计算机处理数据时处理器通过数据总线一次存取、加工和传送的数据位数称为字，这时字通常由 1 字节或若干字节组成。

④ 字长（Word Length）：指处理器的二进制位数。8 位微处理器的字长为 8 位，16 位微处理器的字由 2 字节构成，32 位微处理器的字则由 4 字节构成。字长是衡量计算机性能的一个重要标志，字长越长，性能越强，精度越高。

（3）小数点不能直接标出，需要按一定方式约定小数点的位置

需要说明的是，在计算机系统开发和应用中会涉及二进制（Binary，B）、八进制（Octal，O 或 Q）、十进制（Decimal，D）和十六进制（Hexdecimal，H）等数制。其中，十六进制最常用、最方便，与二进制的关系简明、直接，并用 0～9 和 A～F（H）代表 0～15（D）。

1.2.2 数的表示方法——原码、反码和补码

为了妥善解决符号数值化的表示与运算问题，设定了机器数 x 的三种编码形式，即原码、反码和补码，分别记为$[x]_{原}$、$[x]_{反}$和$[x]_{补}$。

1. 原码

设 $x=x_1x_2\cdots x_{n-1}$，其中 x_i 为一位二进制数，$i=1, 2, \cdots, n-1$。n 位二进制数

$$[x]_{原}=\begin{cases}0x_1x_2\ldots x_{n-1} & x\geqslant 0\\ 1x_1x_2\ldots x_{n-1} & x\leqslant 0\end{cases}$$

一个数的原码就是数值部分为绝对值，加上用 0 和 1 分别表示数的符号+和−的机器数。

【例 1-1】 x_1=67= +1000011，$[x_1]_{原}$=01000011；x_2=−67= −1000011，$[x_2]_{原}$=11000011。

在原码表示法中，根据定义，数 0 的原码有两种不同形式（设字长为 8 位）：

$[+0]_{原}$=00000000　　$[-0]_{原}$=10000000

原码表示简单易懂，而且与真值的转换方便。但原码表示的数不便于计算机运算，因为在两原码数运算时，先要判断它们的符号，再决定用加法还是用减法，导致机器的结构相应地复杂化或增加机器的运算时间。为解决上述问题，引入反码和补码表示法。

2. 反码

设 $x=x_1x_2\cdots x_{n-1}$，其中 x_i 为一位二进制数，$i=1, 2, \cdots, n-1$。定义

電子工業出版社

$$[x]_{反}=\begin{cases}0x_1x_2\ldots x_{n-1} & x\geqslant 0\\ 1\overline{x}_1\overline{x}_2\ldots\overline{x}_{n-1} & x\leqslant 0\end{cases}$$

式中，

$$\overline{x}_i=\begin{cases}0 & x_i=1\\ 1 & x_i=0\end{cases}$$

根据定义，正数的反码与原码相同；对于负数，反码的符号位为 1，其余位为数值位按位取反。

【例 1-2】 x_1=83=+1010011，$[x_1]_{反}$=01010011；x_2=–83=–1010011，$[x_2]_{反}$=10101100。

在反码表示法中，根据定义，数 0 的反码也有两种不同形式（设字长为 8 位）：

$[+0]_{反}$=00000000　　$[-0]_{反}$=11111111

3．补码

引入补码的概念，一方面是为了解决原码、反码存在的一些问题，另一方面是为了将计算机中的加、减运算简化为单纯的加法运算，这种运算上的特性与补码本身的定义有关。

（1）同余的概念和补码

设有两个数 a=17，b=27，若用 10 去除 a 和 b，则它们的余数均为 7，我们称 17 和 27 在以 10 为模时是同余的，并记为 17=27 (mod 10)。

或者说，17 和 27 在以 10 为模时是相等的。此处的模是一个计量系统所能表示的最大量程（或一个计量单位称为模或模数）。

由同余概念，不难得出 $a+M=a$ (mod M)，$a+2M=a$ (mod M)，因此当 a 为负数时，如 $a=-4$，在以 10 为模时，有

–4+10 =–4 (mod 10)　　即　6=–4 (mod 10)

上式说明，在以 10 为模时，–4 与+6 是相等的。我们称+6 为–4 的补码，或者说+6 与–4 对模 10 来说互为补数。有了补码的概念，就可将减法转化为加法（加补码）来进行。如：

7–4=7+6 (mod 10)

该式说明，在以 10 为模时，7 减 4 可以通过 7 加–4 的补码 6 来进行，而所得结果是相同的（只要在加补码 6 时，将所产生的进位舍弃即可，这正是以 10 为模的意思）。

现在把同余和补码的概念推广到二进制数。设计算机字长为 n 位，那么其模为 2^n，即

$$\underbrace{1\overbrace{0\quad 0\cdots 0}^{n\text{ 个 }0}}_{n+1\text{ 位}}$$

也就是说，对于字长 n 位的机器，2^n 在机器中仅能以 n 个 0 表示。或者说，2^n 和 0 在机器中表示的形式是一样的。

（2）补码求法

设 $x=x_1x_2\cdots x_{n-1}$，其中 x_i 为一位二进制数，i=1, 2, …, n–1。定义

$$[x]_{补}=\begin{cases}0x_1x_2\ldots x_{n-1} & x\geqslant 0\\ 1\overline{x}_1\overline{x}_2\ldots\overline{x}_{n-1}+1 & x\leqslant 0\end{cases}$$

電子工業出版社

或写成

$$[x]_{补}=\begin{cases}x & 0\leqslant x\leqslant 2^{n-1}-1\\ 2^{n}+x & -2^{n-1}\leqslant x\leqslant 0\ \ (\mathrm{mod}\ 2^{n})\end{cases}$$

① 正数的补码。$[x]_{补}=[x]_{原}=[x]_{反}$。

【例 1-3】 $[+127]_{原}=[+127]_{反}=[+127]_{补}=\underbrace{01111111}_{\text{字长为8位}}$。

② 负数的补码。先求负数的反码，然后在反码最低一位上加 1 就得到该负数的补码。

还可以用形式上更简便的方法求得负数 x 的补码：符号位为 1，将原数值中最右边一个 1 及其后面的 0 保持不变，而最右一个 1 以左的各位按位取反。

【例 1-4】 $x_1=-0011000$，$[x_1]_{补}=11101000$；$x_2=-1000000$，$[x_2]_{补}=11000000$。

根据定义，数 0 的补码仅有一种形式：$[+0]_{补}=[-0]_{补}=\underbrace{00\cdots0}_{n位}$。

若已知一个负数的补码，再取一次补，则 $\{[x]_{补}\}_{补}=[x]_{原}$。

【例 1-5】 设$[x]_{补}$=10010111，求 x 的真值。

$$\begin{array}{r}11101000\\ +\qquad\ 1\\ \hline 11101001\end{array}$$

所以，$[x]_{原}$=11101001，x=−1101001=−105。

当已知 8 位二进制数补码符号位为 1 时，表示该数为负数。此时要注意，其余几位不是该二进制的数值，一定要把它们按位取反，且在最低位加 1，才是该二进制数的值（真值）。

十进制数及其对应的 8 位二进制数的原码、反码和补码表示如表 1-1 所示。

表 1-1　十进制数及其对应的 8 位二进制数的表示方法

十进制数	二进制数	原　码	反　码	补　码
+0	+0000000	00000000	00000000	00000000
+1	+0000001	00000001	00000001	00000001
+2	+0000010	00000010	00000010	00000010
…	…	…	…	…
+126	+1111110	01111110	01111110	01111110
+127	+1111111	01111111	01111111	01111111
−0	−0000000	10000000	11111111	00000000
−1	−0000001	10000001	11111110	11111111
−2	−0000010	10000010	11111101	11111110
…	…	…	…	…
−126	−1111110	11111110	10000001	10000010
−127	−1111111	11111111	10000000	10000001
−128	−10000000	无法表示	无法表示	10000000

小结：

① 有符号数三种编码的最高位都是符号位。符号位为 0，表示真值为正数，其余位为真值；符号位为 1，表示真值为负数，其余位除原码外，不再是真值。对于反码，只需按位

电子工业出版社

取反，便是真值；对于补码，还需按位取反加 1，才是真值。

② 对正数，三种编码是一样的，即$[x]_{原}=[x]_{反}=[x]_{补}$；对于负数，三种编码就不同了。所以说，原码、反码和补码的实质是用来解决负数在机器中的表示而引入的编码方法。

③ 8 位二进制数原码、反码和补码所能表示的数值范围不完全相同，它们分别是−127～+127，−127～+127 和−128～+127。其中对 0 的表示也不尽相同，原码和反码有两种表示方法，补码只有一种表示方法。

④ 当计算机采用不同的码制时，运算器和控制器的结构将不同。采用原码形式的计算机称为原码机。类似地，有反码机和补码机。小型计算机和微型计算机大多为补码机（实际上原码机与反码机已不采用了）。

1.2.3 补码的运算

从以上讨论可知，在微型计算机中，有符号数以补码的形式在机器中存储和运算。这主要是因为补码的加、减运算比原码简单；符号位与数值位一起参加运算，并能自动获得正确结果。

设 x 和 y 是两个正数，可以证明，两个数和的补码等于两个数补码的和：

$$[x+y]_{补}=2^n+(x+y)=(2^n+x)+(2^n+y)=[x]_{补}+[y]_{补}$$

同样可以证明，该两数差的补码等于被减数的补码与减数负值的补码（或称为求补）之和。

$$[x-y]_{补}=2^n+(x-y)=2^n+x+2^n+(-y)=[x]_{补}+[-y]_{补}$$

上式说明，在补码运算中，两数差的运算可简化为单纯的加法运算。

x 的补码直接由补码定义可得到，而$[-y]_{补}$可通过对$[y]_{补}$“连同符号位在内一起取反加 1”得到（称为“求补”）。

【例 1-6】 $[y]_{补}=00000100$，$[-y]_{补}=11111100$。

【例 1-7】 计算 $x-y$。x，y 均为正数，且 $x>y$。设 $x=122$，$y=37$，字长 $n=8$。

解：

十进制计算	二进制补码计算
122	01111010 = $[x]_{补}$
+ 37	+ 11011011 = $[y]_{补}$
85	101010101

进位自动舍去 ↑└——— 符号位为 0，表示正数

求得真值为正数$(01010101)_2=85$。

【例 1-8】 计算 $x-y$。x 和 y 均为正数，且 $x<y$。设 $x=64$，$y=65$，字长 $n=8$。

解：

十进制计算	二进制补码计算	
64	01000000 = $[x]_{补}$	
− 65	+ 10111111 = $[-y]_{补}$	
−1	11111111	符号位为 1，表示负数

求真值：$(11111111)_{补}=\{(11111111)_{补}\}_{补}=-(00000001)=-1$。

【例 1-9】 计算 $x+y$。x，y 均为正数。设 $x=64$，$y=65$，字长 $n=8$。

解：

十进制计算	二进制补码计算	
64	01000000 = $[x]_{补}$	
+ 65	+ 01000001 = $[y]_{补}$	
129	10000001	符号位为 1，表示负数

電子工業出版社

此时，两个正数相加得出负数，显然是错误的。这种情况称为“溢出”。由表 1-1 知，8 位计算机中，由于最高位为符号位，剩下的数值位只有 7 位，表示数的范围是–128～+127。当两个正数相加其和大于 127 或两个负数相减其绝对值之和大于 128 时，就产生了“溢出”，致使结果出错。推广到字长 n 位的符号数，最高位为符号位，n–1 位表示数值。能表示的最大值为 $2^{n-1}-1$（即 n–1 个 1）。当运算结果超过此值，就会产生“溢出”。

小结：

（1）补码运算时，参加运算的两个数均为补码，结果也是补码，需转换才能得到真值。

（2）可以将减法变为加法运算，以简化硬件设计。

（3）运算时：① 符号位与数值位一起参加运算；② 符号位产生的进位可舍弃；③ 要保证运算结果不超过补码所能表示的数的最大范围，否则将产生“溢出”错误。为此，在计算机中设有专门电路用以判断运算结果是否产生溢出，并设置溢出标志告知本次运算的结果的状态。

（4）无符号数和有符号数的加法运算可用同一电路完成。

1.2.4 定点数与浮点数

在计算机中，不仅要处理整数运算，也要处理小数运算，如何处理小数点位置是十分重要的，通常用定点法和浮点法来表示小数点的位置。

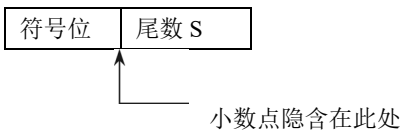
（1）定点表示法

定点表示法就是小数点位置在数中固定不变。例如：

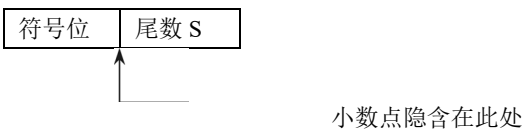
1.100101　　0.100101　　110.0101

一般来说，小数点位置固定在哪个位置上并无限制，但为了使用简便，在计算机中有两种常用的定点数。

① 定点纯小数。把小数点固定在最高数值位左边，在小数点左边设有一位符号位，小数点本身不占位，其格式为：

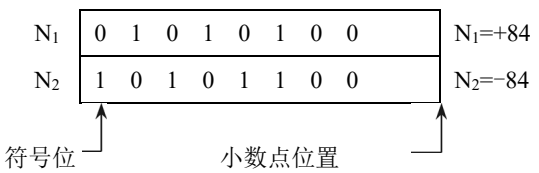


② 定点纯整数。把小数点固定在最低数值位右边，最高位仍为符号位，小数点本身不占位，其格式为：



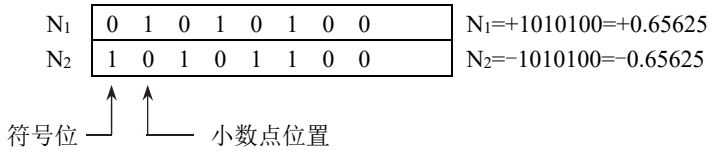
定点数的这两种表示法在计算机中均有采用。但对一台机器而言，采用哪种方法需事先约定。

【例 1-10】 有如下两个 8 位二进制数：



電子工業出版社

【例 1-11】 有例 1-10 中同样两个数，但小数点位置不同，则



从上面两个例子可以看出，定点整数和定点小数在格式上毫无差别，这是因为定点数的小数点是隐含的，但它们的真值不相同。此外，相同位数、不同码制的定点数表示的数的范围也不相同。

同一台计算机可处理的整数类型往往有很多种，以目前广泛使用的 Pentium 机为例，除了无符号整数，它还有 3 种类型的整数，如表 1-2 所示。

表 1-2　Pentium 处理器的 3 种整数类型

整数类型	数值范围	精　度	格　式
16 位整数	–32768～32767	二进制 16 位	16 个二进制位，补码表示
短整数	-2^{31}～$2^{31}-1$	二进制 32 位	32 个二进制位，补码表示
长整数	-2^{63}～$2^{63}-1$	二进制 64 位	64 个二进制位，补码表示

由于定点数是将小数点的位置固定，因此运算起来很不方便。一方面，它要求对所有原始数据要用比例因子化成小数或整数，算出结果又要用比例因子折算成真值；另一方面，这种方法所表示的数的范围小、精度低。所以有必要引入另一表示方法——浮点表示法。

（2）浮点表示法

为了在位数有限的前提下，尽量扩大数的表示范围，同时又保持数的有效精度，计算机往往采用浮点数表示数值。在浮点表示初期，把一个数通过改变小数点位置表示成 2 的 p 次幂和绝对值小于 1 的数 S 相乘的形式：

$$N = 2^p S$$

式中，N 称为浮点数或实数；S 是 N 的尾数，它是数值的有效数字部分，通常用带符号的定点小数表示，一般用原码表示；2 是 N 的底数，是该进位计数制的基数，在计算机中不出现，是隐含的；p 是指数，称为阶码，通常为带符号整数，一般用补码表示。阶码 p 的大小决定了数的范围，尾数 S 的长短则规定了数的有效数字的位数（精度）。在计算机中，p 和 S 均为二进制数。

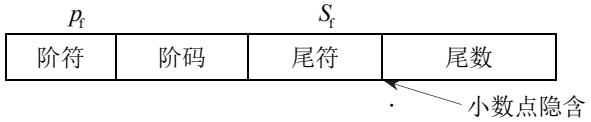
【例 1-12】 试将以下数表示成浮点数形式。

$$1011.1101\text{B} = (0.10111101\text{B})\times 2^4 = (0.10111101)\times 2^{100}\text{B}$$

$$0.00101101\text{B} = (0.101101\text{B})\times 2^{-2} = (0.101101)\times 2^{-10}\text{B}$$

$$-111001010\text{B} = (-0.111001010\text{B})\times 2^9 = (-0.111001010)\times 2^{1001}\text{B}$$

浮点数在机器中的一种表示形式如下：



也就是说，若要在机器中表示一个浮点数，则阶码和尾数要分别表示，且都有自己的符号位。

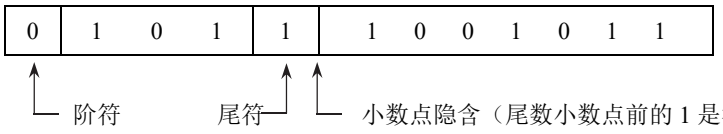
電子工業出版社

通常，用一位二进制数 p_f 表示阶码的符号位。当 $p_f=0$ 时，表示阶码为正；当 $p_f=1$ 时，表示阶码为负。同样，用一位二进制数 S_f 表示尾数的符号位。当 $S_f=0$ 时，尾数为正；当 $S_f=1$ 时，尾数为负。

若浮点数中 $1/2 \leqslant S < 1$，则称该浮点数为规格化了的浮点数。

【例 1-13】 $(-18.75)_{10}=(-10010.11)_2=(-0.1001011)\times 2^{+101}$。

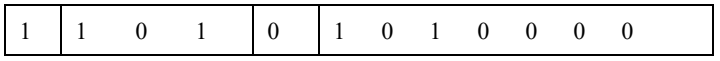
假定尾数用 8 位二进制数表示，阶码用 4 位二进制数表示，且均含符号位，尾数用原码表示，则有



或表示成$(1.1001011)\times(10)^{0101}$。

【例 1-14】 $(0.078125)_{10}=(0.000101)_2=(0.101)\times 2^{-11}$。

同上例



阶码用补码表示，尾数用原码表示，或表示成$(0.1010000)\times(10)^{1101}$。

最后介绍浮点数表示数的范围，若浮点数采用下列格式：

阶符	阶码	尾符	尾数
p_f（1 位）	p 位	S_f（1 位）	S 位

浮点数表示数的范围是 $-2^{2^p-1}(1-2^{-s}) \sim 2^{2^p-1}(1-2^{-s})$。

由于指数可以选用不同的编码（原码、补码等），尾数的格式和小数点位置也可以有不同规定，因此浮点数的表示方法不是唯一的。不同计算机有不同的规定，这就引起了相互间数据格式的不兼容性。为此，美国电气与电子工程师协会（IEEE）制定了有关的工业标准，已被众多计算机制造厂商所采用。

浮点数的长度可以是 32 位、64 位甚至更长，位数越多，可表示的数的范围越大，精度也越高。

以当今流行的 Pentium 处理器中的浮点数为例。Pentium 处理器中的浮点数格式完全符合 IEEE 标准，它表示成如下形式：

$$(-1)^{S_f}\,2^{p}\,(b_0.b_1b_2b_3\cdots b_{s-1})$$

小数点隐含

式中，$(-1)^{S_f}$ 是该数的符号位，S_f=0 或 1，0 表示正，1 表示负；p 是指数，它是一个带偏移量的整数，表示成无符号整数；$(b_0.b_1b_2b_3\cdots b_{S-1})$ 是尾数，其中 $b_i=0$或1$(i=0,1,2,\cdots,S-1)$；S 是尾数的长度，它表示尾数共 S 位。

表 1-3 为 Pentium 处理器中三种类型浮点数的格式。Pentium 处理器浮点数的尾数最高位 b_0 总是 1，且和小数点一样隐含，在机器中不表示。

【例 1-15】 将十进制数 178.125 表示成单精度浮点数。

首先，将十进制数 178.125 表示成二进制实数：$(178.125)_{10}=(10110010.001)_2$。

然后，将二进制实数表示成规格化形式：$10110010.001=1.0110010001\times 2^7$。

電子工業出版社

表 1-3　Pentium 处理器中三种类型浮点数的格式

参　　数	单精度浮点数	双精度浮点数	扩充精度浮点数
浮点数长度（字长）	32	64	80
尾数长度 S	24	53	64
符号位 S_f	1	1	1
指数 p 的长度	8	11	15
最大指数	+127	+1023	+16383
最小指数	-126	-1022	-16382
指数的偏移量	+127	+1023	+16383
可表示的偏移量	10^{-38}～10^{38}	10^{-308}～10^{308}	

而 $1.0110010001\times2^{7}=1\,0110010001\times2^{111}$

↑ 小数点隐含

因为指数等于 7，加上偏移量 127 后，p=7+127=134=$(10000110)_2$。所以，178.125 的单精度浮点数表示为：

0	10000110	01100100010000000000000
S_f	p	$b_1b_2b_3$ … b_{23}

【例 1-16】单精度浮点数 001111110 10110000000000000000000 的十进制数值是多少？

先计算指数。因为 p=$(01111110)_2$=$(126)_{10}$，所以指数为 126−127=−1。

再计算尾数。规格化的尾数是+1.1011。

所以，该浮点数的数值为 $+1.1011\times2^{-1}=+0.11011=(+0.84375)_{10}$。

【例 1-17】 下面 4 个等式中，等号左面是二进制实数，等号右面是其等值的单精度浮点数表示。

$0.11010001\times2^{10110}$ = 0 10010100 10100010000000000000000

$-0.11010001\times2^{10110}$ = 1 10010100 10100010000000000000000

$0.11010001\times2^{-10010}$ = 0 01101100 10100010000000000000000

$-0.11010001\times2^{-10010}$ = 1 01101100 10100010000000000000000

需要指出的是，任意位精度浮点数可由定点处理器计算生成，而上文所述的属于机器精度浮点数，其精度由特定格式规定，后者的应用更广泛。

相对于整数（定点数）的表示形式，浮点表示法所表示的数据范围大、精度高，但同时增加了运算的复杂度和时间开销。因而众多计算机系统中都将浮点运算集成为单独硬件部件——浮点运算单元（FPU，Floating-Point Unit），以增强浮点处理的能力。Intel 公司在 1980 年发布了 8087 数学协处理器（浮点运算器件），该协处理器只能与 8086/8088 配合使用，主要负责完成浮点运算并将运算结果传输给 CPU，其数据类型为 IEEE 标准格式。此后，Intel 公司还相继推出了 80287 和 80387 协处理器。1989 年发布的 Intel 486DX 中，已经将浮点运算部件做在 CPU 内部，而不再是作为一个选件。

1.2.5　BCD 码及其十进制调整

二进制数具有很多优点，所以到目前为止在计算机内部多采用二进制运算。但是，二进制数书写起来很长，读起来也不方便。考虑到人机交互的需求，通常，数在送入机器前

電子工業出版社

仍采用十进制编码，运算结果也以十进制数输出。这就要求在输入时将十进制数转换成二进制数，输出时将二进制数转换成十进制数，这项工作一般由机器自动完成。因而产生了一种适合于十进制数的二进制代码的特殊形式，即用 4 位二进制数来表示一位十进制数，简称 BCD（Binary Coded Decimal）码。

要表示一个十进制数码，至少需要 4 位二进制数组合，其编码的选择有很多种方法，其中常用的 BCD 码如表 1-4 所示。最常用的是用 0～9 各数字所对应的二进制数作为代码，称为 8421 BCD（简称 NBCD）码。

表 1-4　常用 BCD 码编码表

十进制数	8421	2421	余 3 码	格 雷 码
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0010	0101	0111
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1100
6	0110	1100	1001	1101
7	0111	1101	1010	1111
8	1000	1110	1011	1110
9	1001	1111	1100	1010

从表 1-4 可见，8421 码与十进制数的对应关系最直观，转换十分简单，只需将十进制数的各位数字用与其对应的一组 4 位二进制数代替即可。下文所述 BCD 码仅指 8421BCD 码。

【例 1-18】 将十进制数 83.6 转换成 BCD 码。

$$(83.6)_{10}=(1000\ 0011.0110)_{BCD}$$

【例 1-19】 将 BCD 码 1001 0111.0100 0010 转换成十进制数。

$$(1001\ 0111.0100\ 0010)_{BCD}=(97.42)_{10}$$

注意，BCD 码与真正的纯二进制数是不同的，貌似二进制数，实为十进制数。由于 BCD 码只是将每个十进制数用 4 位二进制数来表示，而每组 4 位二进制数之间仍然是逢十进一的关系。因此，如果将这种 BCD 码直接交由计算机去运算，由于计算机总是将其作为二进制数来处理，结果可能会出错。

【例 1-20】 求 BCD 码的 7+6=?

$$\begin{array}{r} 0111 \\ +\ 0110 \\ \hline 1101 \end{array}$$

1101 为非法 BCD 码，运算结果出错，7+6 的正确 BCD 码应为$(00010011)_{BCD}$。

为了得到正确的 BCD 码的运算结果，必须对二进制数的运算结果进行调整，使之符合十进制数的运算和进位规律。这种调整称为十进制调整，其规则如下：

① 若两个 BCD 数相加的结果大于 9（1001），则应作“加 6（0110）修正”。

② 若两个 BCD 数相加的结果在本位上并不大于 9（1001），但产生了进位，相当于十进制数运算大于等于 16，则也应“加 6（0110）修正”。

【例 1-21】 求 BCD 码 54+48=?

電子工業出版社

```
    01010100
+   01001000
-------------
    10011100
+       0110      加 6 修正
-------------
    10100010
+   0110          高 4 位加 6 修正
-------------
000100000010
```

低 4 位之和为 1100，应作加 6 修正，修正后高 4 位又为 1010，还应再作一次加 6 修正。故最终结果为 102。

注意：实际上，在 Intel 兼容计算机中有十进制调整指令 DAA（DAS），无论对加法或减法，机器能按照规则自动进行调整（加法时，“加 6 修正”；减法时，“减 6 修正”）。

BCD 码在计算机中有两种表示方法，一种是用 1 字节表示两位十进制数，称为压缩 BCD，如$(56)_{10}=(01010110)_{BCD}$；另一种称为非压缩 BCD，用 8 位二进制数表示一位十进制数，如$(56)_{10}=(0000010100000110)_{BCD}$。前者占内存少，利用率高；后者占内存多，但其运算及调整相对简单。

1.3 计算机中非数值数据的信息表示

计算机除了能对数值信息进行处理（主要是各种数学运算）之外，对于诸如文字、图形、图像、声音等信息也能进行各种处理，当然它们在计算机内部也必须表示成二进制编码形式，这些统称为非数值数据。

1.3.1 西文信息的表示

西文包括拉丁字母、数字、标点符号及一些特殊符号，它们统称为字符（Character）。

目前，国际上使用的字母、数字和符号的信息编码系统种类很多。经常采用的是美国国家信息交换标准代码 ASCII（American Standard Code for Information Interchange）。该标准制定于 1963 年，目前微型计算机的字符编码都采用 ASCII。

ASCII 是一种 8 位代码，一般最高位可用于奇偶校验，故仅用 7 位码来代表字符信息，如表 1-5 所示。ASCII 字符集共有 128 个字符，其中 34 种起控制作用的称为“功能码”，其余 94 种符号（10 个十进制数码、52 个英文大/小写字母和 32 个专用符号$、+、−、=等）供书写程序和描述命令之用，称为“信息码”。

我国于 1980 年制定了“信息处理交换器的 7 位编码字符集”，即国家标准 GB1988—1980，除了用人民币符号“￥”代替美元符号“$”，其余含义都与 ASCII 相同。

ASCII 字符集适合于编码英文字符，但是对于其他特殊字符或希腊语、俄文等语言的字符无法支持。Unicode 联合会修订了最全面且广泛接受的文字编码标准，当前的 Unicode 标准（7.0 版）的字库包括将近 10 万个字符，支持广泛的语音种类，包括古埃及和巴比伦的语言。Unicode 的“统一字符集”的基本编码采用 32 位来表示单个字符，但是对于常用字符将使用 1～2 字节编码，而不太常用的字符则使用最多 4 字节编码。特别地，UTF-8 将每个字符编码为单字节，且与 ASCII 使用相同的编码值，因而实现与 ASCII 的兼容。

電子工業出版社

表 1-5　ASCII 字符表

低位 / 高位	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1 0001	DLE	DC_1	DC_2	DC_3	DC_4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2 0010	SP	!	“	#	$	%	&	、	(	)	*	+	,	-	.	/
3 0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4 0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5 0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	←
6 0110	、	a	b	c	d	e	F	g	h	i	j	k	l	m	n	o
7 0111	p	q	r	s	t	u	V	w	x	y	z	{	|	}	～	DEL

1.3.2　中文信息的表示

中文的基本组成单位是汉字，它们也属于字符。相比于西文字符，汉字具有数量大、字形复杂、同音字多等特点，这就给汉字在计算机内部的表示与处理、汉字的传输与交换、汉字的输入/输出等带来了一系列的问题。为此，我国于 1981 年颁布了“国家标准信息交换用汉字编码基本字符集（GB2312—1980）”。该标准规定，一个汉字用 2 字节（256×256=65536 种状态）编码，同时用每字节的最高位来区分是汉字编码还是 ASCII 字符编码，这样每字节只用低 7 位，这就是所谓双 7 位汉字编码（128×128=16384 种状态），称为汉字交换码（又称为国标码），其格式如图 1-13 所示。

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	×	×	×	×	×	×	×	0	×	×	×	×	×	×	×

图 1-13　国标码格式

目前，许多机器为了在内部能区分汉字与 ASCII 字符，把 2 字节汉字的国标码的每字节的最高位置 1，这样就形成了汉字的另一种编码，称为汉字机内码（内码）。若已知国标码，则机内码唯一确定。方法是，机内码的每字节为原国标码每字节加 80H。机内码用于统一不同系统所使用的不同汉字输入码，各种不同汉字输入码进入系统后，一律转换为机内码，使不同系统内的汉字信息可以相互转换。

汉字输入方法很多，如区位、拼音、五笔字型等数百种。其中最优者应具有易学习、易记忆、效率高（击键次数少）、重码少和容量大等特点。不同输入法有自己的编码方案，不同输入法所采用的汉字编码统称为输入码。输入码进入机器后，必须转换为机内码。

传统的汉字输出是先用汉字字形码（一种用点阵表示汉字字形的编码）把汉字按字形排列或点阵，常用点阵有 16×16、24×24、32×32 或更高。汉字字形点阵的信息量很大，占用存储空间也较大。不同字体、字号的汉字字形构成字体，通常都存储在硬盘上，只有当

电子工业出版社

要显示输出时，才去检索得到欲输出的字形，典型的输出字形可用矢量法、True Type 等。

习 题 1

1-1　简述计算机和微型计算机经过了哪些主要发展阶段，通过上网查阅资料，获取最新的主流微型计算机的相关信息。

1-2　什么叫微处理器？什么叫微型计算机？什么叫微型计算机系统？

1-3　通用微型计算机硬件系统结构是怎样的?请用示意图表示，并说明各部分的作用。

1-4　冯·诺依曼计算机的基本设计思想是什么？

1-5　简述 RISC 和 CISC 微处理器结构的特性，并通过查阅资料，给出典型实例加以说明。

1-6　典型微机为何选择总线结构？请简述总线结构的分类及各自的特点。

1-7　Intel 公司最新的通用处理器是什么型号？请查阅相关资料，说明其主要特点。

1-8　什么是指令？什么是程序？

1-9　试用示意图说明内存储器的结构和读/写操作。

1-10　简述衡量通用微型计算机的技术指标。

1-11　写出下列机器数的真值。

（1）01101110　　（2）10001101

1-12　写出下列二进制数的原码、反码和补码（设字长为 8 位）。

（1）+010111　　（2）−101000

1-13　当下列各二进制数分别代表原码、反码和补码时，其等效的十进制数值为多少?

（1）00001110　　（2）10000001

1-14　已知 x_1=+0010100，y_1=+0100001，x_2=−0010100，y_2=−0100001，试计算下列各式（设字长为 8 位）。

（1）$[x_1+y_1]_{补}$　　（2）$[x_1-y_2]_{补}$　　（3）$[x_2-y_2]_{补}$

（4）$[x_2+y_2]_{补}$　　（5）$[x_1+2y_2]_{补}$　　（6）$[x_2+y_2/8]_{补}$

1-15　用补码来完成下列计算，并判断有无“溢出”产生（设字长为 8 位）。

（1）85+60　　（2）–85+60　　（3）85–60　　（4）–85–60

1-16　在微型计算机中存放两个补码数，试用补码加法完成下列计算，并判断有无“溢出”产生。

（1）$[x]_{补}+[y]_{补}$=01001010+01100001　　（2）$[x]_{补}-[y]_{补}$=01101100−01010110

1-17　试将下列各数转换成 BCD 码。

（1）$(30)_{10}$　　（2）$(127)_{10}$　　（3）00100010B　　（4）74H

1-18　试写出下列字符的 ASCII 值：9，　*，　=，　$，　!。

1-19　若加上偶校验，题 1-18 中字符的 ASCII 又是什么?

1-20　试将下列数表示成浮点规格化数。设阶码（含阶符）为 4 位，尾数（含数符）为 8 位。

（1）69.57　　（2）−0.3125

1-21　试将 1-20 题中的数表示成单精度浮点数。

1-22　下面是 Pentium 处理器的一个单精度浮点数（32 位）和一个双精度浮点数（64 位），这两个浮点数的实际数值为多少？

（1）01000010101100000000000000000000　　（2）406CD25179FCED82H

電子工業出版社

第 2 章　微处理器

微处理器是微型计算机的核心部件，负责完成数据处理和过程控制，广泛应用于各种民用、航空航天、军用的电子和计算机系统中。1971 年，Intel 公司推出了世界上第一片微处理器 4004，其后伴随着集成电路制造工艺和设计开发技术的进步，微处理器在集成度、工作速度、运算性能等多个维度上不断提升，并延展出诸如数字化智能设备、超高速巨型计算机等众多新型应用场景。本章将以 Intel 80x86/Pentium 系列微处理器为例介绍微处理器的结构、引脚功能、工作时序和典型应用系统。

2.1　微处理器概述

前文已述及，从指令集的结构来区分，微处理器可分为 RISC 和 CISC 处理器，而从应用领域来区分，微处理器可分为通用微处理器、嵌入式微处理器（数字信号处理器）和微控制器。通用微处理器主要用于运行通用软件，追求总体的高性能，通常配备功能完备、复杂的操作系统，典型代表为 Intel 的 80x86 系列、AMD 的 K 系列等；嵌入式微处理器主要用于面向特定应用领域，强调处理特定问题的高性能，通常配备轻量级的操作系统，主要用于消费类电子产品，典型代表为 ARM 系列、MIPS、PowerPC、DSP 等；微控制器实现系统的控制功能，性能和价格相对较低，主要用于汽车、自动机械等的自控设备，典型代表为单片机 MCU。本章将会以 Intel 80x86 系列处理器为例详细说明微处理器的特性，下面将简单介绍其他几类常用的微处理器并给出 64 位微处理器的最新发展。

（1）ARM（Advanced RISC Machine）

ARM 微处理器是目前在嵌入式系统应用最为广泛的处理器，可分为 ARM7、ARM9、ARM9E、ARM10E、ARM11、Cortex、SecurCore、OptimoDE Data Engines、Intel 的 Xscale、Intel 的 StrongARM 等系列。其中除了 ARM7 系列和 Cortex-M0、Cortex-M23 采用冯 • 诺依曼体系结构，其余均为哈佛结构。作为 RISC 处理器的典型代表，ARM 微处理器具有以下几点主要特征：

① 31 个无区别使用的通用寄存器，均可存放数据或地址，可为所有的数据操作提供快速的局部存储访问。

② 在符合 RISC 指令长度固定的特性基础上，优化调整特定指令的周期，并给出 32 位指令集的 16 位压缩形式 Thumb 指令集，提高了代码密度。

③ 在输入寄存器被某条指令使用前，内嵌桶型移位寄存器可处理该寄存器中的数据，扩展了指令功能，优化内核性能。

④ 支持用户、快中断、中断、管理、中止、系统和未定义等 7 种处理器模式，除了用户模式外，其余的均为特权模式。

⑤ 具有片上总线 AMBA（Advanced Micro-controller Bus Architecture）。AMBA 定义了 3 组总线：先进高性能总线 AHB（Advanced High performance Bus），先进系统总线 ASB（Advanced System Bus），先进外围总线 APB（Advanced Peripheral Bus）。AMBA 可以方便

電子工業出版社

地扩充各种处理器及 I/O，可以把 DSP、其他处理器和 I/O（如 UART、定时器和接口等）都集成在一块芯片中。

⑥ 指令执行采用 3 级/5 级流水线技术。

⑦ 采用 Load-Store 结构，即 CPU 并不会对内存中的数据进行操作，所有的计算都要求在寄存器中完成，而寄存器和内存的通信则由单独的指令来完成。

⑧ ARM 处理器主要应用的领域包括：无线、消费电子和图像应用的开放平台；存储、自动化、工业和网络应用的嵌入式实时系统；智能卡和 SIM 卡的安全应用等。

（2）DSP（Digital Signal Processor）

DSP 芯片是适合于进行数字信号处理的微处理器，主要应用是实时快速地实现各种数字信号处理算法，具有运算速度快、可程控、可预见性、精确度高及接口灵活等优点，也有功耗大、系统复杂、应用的频率范围受限等缺点，典型产品为 TI 公司的 TMS320 系列。

由于面向实时信号处理任务，DSP 作为一种微处理器，其设计的出发点和通用 CPU 以及 MCU 等处理器不同，其设计核心是算法的高效实现，因而在体系结构方面 DSP 具有如下的特性。

DSP 具有独立的硬件乘法器，乘法指令可以在单周期内完成，使得卷积、数字滤波、FFT、相关运算、矩阵运算等算法中的大量乘法运算速度加快。

DSP 通常包含专门的地址产生器，能生成信号处理算法需要的特殊寻址，如循环寻址和位翻转寻址，其中循环寻址对应于流水 FIR 滤波算法，位翻转寻址对应于 FFT 算法。

DSP 具有多总线结构，程序空间与数据存储空间分开，各自独立的地址总线和数据总线（哈佛结构），取指令和读数据可以同时进行。

采用流水线技术，每条指令的执行可分为取指令、译码、取数、执行等若干步骤，由片内多个功能单元完成，相当于多条指令并行执行（RISC 特征），显著提高了运算速度。

片内具有快速的 RAM，通常可通过独立的数据总线在两块 RAM 中同时访问。

快速的中断处理和硬件 I/O 支持。

VLIW（超长指令字）技术使用静态调度方法，使用宽的并行指令独立控制各个功能单元，可以进一步优化执行速度。

（3）MCU（Micro Controller Unit）

单片机 MCU 是将 CPU、RAM、ROM、多个 I/O 接口和中断系统、定时器/计数器等功能集成到单个芯片的微型计算机系统，主要应用在工业控制领域。单片机的主要特点如下。

① 片内寄存器分为工作寄存器组和特殊功能寄存器，可实现数据访问和多种 I/O 接口的数据管理。

② 片内有全双工串行通信接口，可较为方便地组成分布式控制系统。

③ 片内有独立的位处理器，又称为布尔处理器，具有相应的累加器和可按位寻址的 RAM 区、特殊功能寄存器，并设有专门的位指令，方便实现位控功能。

④ 具有面向控制、结构简单、体积小、可靠性高的特点，由于自带较多的 I/O 接口，可以直接对 I/O 进行算术、逻辑和位操作，指令简单而丰富，控制能力强。而且 I/O 端口多采用分时复用的方式，整体芯片体积紧凑，且易于采取屏蔽措施，适合在恶劣环境下工作。

⑤ 具有低电压、低功耗、较高性价比的特点，通常可工作在 2.2 V 或 1.2 V 等低电压，由于 MCU 构成的硬件结构简单、开发周期短、控制能力强、可靠性高，在同等功能的要求下，基于 MCU 开发的控制系统比其他类型微处理器开发的相应系统具有价格上的优势。

電子工業出版社

上文给出了除 Intel 的微处理器之外的几种常用处理器的简单介绍，下面将说明目前广泛使用的 64 位处理器和多核处理器的特性。

64 位 CPU 是指一次处理数据的宽度为 64 位的处理器。它们可以使用 64 位格式存储数据，并可以对 64 位操作数执行数学运算操作。目前，市场上可以实现 64 位计算的与 Intel 兼容的处理器主要有 3 种：① Intel IA-64，基于安腾 2 处理器，不兼容 32 位应用；② Intel EM64T，基于 Xeon DP “Nocona”和 MP 处理器，兼容 32 位应用；③ AMD AMD64，基于 Opteron 处理器，兼容 32 位应用。

由于 80x86 系列也包括 16 位处理器，现在 Intel 公司把 32 位的 x86-32 称为 IA-32，全名为 Intel Architecture 32-bit。x86-64，简称 x64，是 64 位 CPU 架构及其相关指令集的一种，也是 Intel x86 架构的延伸产品。x64 最先由 AMD 设计，推出时被称为 AMD64，其后也为 Intel 公司所采用，现在 Intel 称之为 Intel 64，也称为 EM64T。

Intel 公司之前已经在 Itanium 处理器上使用了 64 位 IA-64 技术，虽然 EM64T 也是 64 位，但两者并不相容，即 IA-64 的软件不能直接在 EM64T 上运行。EM64T 是 IA-32 指令集的延伸，而 IA-64 则是另一种独立的架构，没有任何 IA-32 的影子。虽然 IA-64 可通过模拟方式来执行 IA-32 的指令，但指令需要转换才能在 IA-64 上执行，导致速度变慢。

EM64T 是 Intel IA-32 架构的扩展，即 IA-32E（Intel Architecture-32 Extension）。IA-32 处理器通过附加 EM64T 技术，可在兼容 IA-32 软件的情况下允许软件利用更多的内存地址空间，并且允许软件进行 32 位线性地址写入。EM64T 特别强调对 32 位和 64 位的兼容性。

目前，Intel 支持 64 位技术的 CPU 有 Nocona 核心的 Xeon 系列、Prescott 2M 核心的 P4 EE 系列等。AMD 支持 64 位技术的 CPU 有 Athlon 64 系列、Athlon FX 系列和 Opteron 系列等。

微处理器的整体性能提高大都来源于处理器频率的大幅增加（在 1983 至 2002 年间，从 5 MHz 增加到 3 GHz）和制造工艺技术的进步，还有一部分改进来源于每周期所执行指令数的增加（流水线技术的不断改进）。不过，从 2002 年开始，随着功耗和散热量的不断增加，这种通过提高处理器频率改进性能的方法，已经开始显露出局限性。解决的思路是使用多核处理器（Chip Multi-Processor，CMP），通过将多个内核整合在一起，每个内核能够在较低的频率下运行，从而各单内核的功耗下降，由此得到的多核处理器性能大幅超过单核，能够显著地增强计算机的处理能力和计算资源，提供更好的响应能力、更大的多线程吞吐量，并把并行计算的优势运用到分配了合理线程的主流应用软件中。

IBM 公司在 2001 年推出了基于双核心的 POWER4 处理器，随后 Sun 和 HP 公司都先后推出了基于双核架构的 UltraSPARC 和 PA-RISC 芯片，但此时双核心处理器架构只应用在高端的 RISC 服务器领域，直到 2006 年，AMD 和 Intel 公司相继推出自己的双核处理器，双核才真正走入了主流的 x86 领域。2008 年，Intel 公司推出代号为“Nehalem”的全新微体系结构，目前广泛用于 PC 领域的 Intel 酷睿 i7 正是基于该微结构的。2010 年，Intel 公司向外公布了其首款超多核心商用芯片产品“Knights Corner”，这款产品借用了 Larrabee 架构和万亿级计算研究计划（Tera-scale Computing Research Program）的技术，可以将 50 个以上的处理器核心集成到一块单芯片中。

多核处理器是处理器发展的必然趋势，无论是移动与嵌入式应用、桌面应用还是服务器应用，都将采用多核架构，其未来应用前景十分广阔。在体系结构、软件、功耗和安全性设计等方面，多核处理器面临着巨大的挑战，当然同时指明了其未来优化的方向。

電子工業出版社

2.2 80x86/Pentium 微处理器的内部结构

2.2.1 8086/8088 CPU 基本结构

1．8086/8088 CPU 结构框图

8086/8088 由两个独立的处理部件组成：执行部件 EU（Execution Unit）和总线接口部件 BIU（Bus Interface Unit）。8086 和 8088 两者的执行部件 EU 完全相同，而总线接口部件 BIU 略有不同。8086 BIU 中的指令队列是 6 字节，“外部数据总线”是 16 位；8088 指令队列只有 4 字节，“外部数据总线”是 8 位，如图 2-1 所示。下面分别介绍 EU 和 BIU。

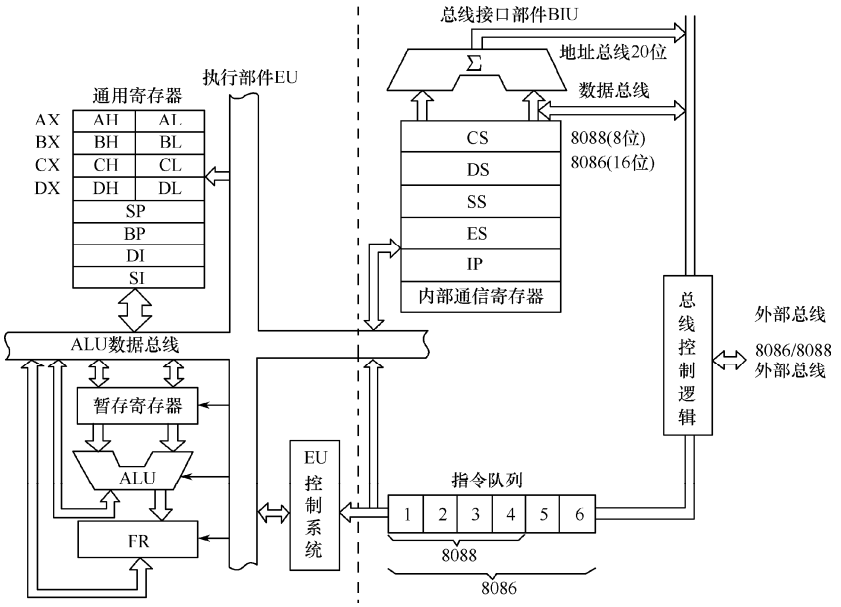


图 2-1 8086/8088 CPU 内部结构

（1）执行部件 EU

EU 包括 8 个 16 位寄存器（通用寄存器 AX、BX、CX、DX；指针寄存器 SP、BP；变址寄存器 SI、DI）、算术逻辑部件 ALU、标志寄存器 FR、暂存寄存器和 EU 控制系统。

EU 负责全部指令的执行，同时向 BIU 输出数据（操作结果），并对寄存器和标志寄存器进行管理。在 ALU 中进行 16 位运算，数据传输和处理均在 EU 控制下进行。

EU 的具体工作过程是：从 BIU 指令队列中取出指令操作码，通过译码电路分析要进行什么操作，发出相应的控制指令，控制数据经过“ALU 数据总线”的流向；如果是运算操作，操作数经暂存寄存器送入 ALU，运算结果经“ALU 数据总线”送到相应寄存器，同时标志寄存器 FR 根据运算结果改变标志位；如果执行指令需从外界取数据，则 EU 向 BIU 发出请求，由 BIU 通过 8086/8088“外部数据总线”访问存储器或外部设备，通过 BIU 的内部通信寄存器向“ALU 数据总线”传送数据。

（2）总线接口部件 BIU

BIU 由 4 个段（Segment）寄存器（CS、SS、DS、ES）、指令指针 IP、内部通信寄存器、指令队列（Queue）、总线控制逻辑和地址加法器组成。

電子工業出版社

BIU 负责执行所有的“外部总线”周期，提供系统总线控制信号，还将段寄存器中的内容与偏移量寄存器中的值送到地址加法器中，形成 20 位存储器的物理（实际）地址。当 EU 执行指令要求与内存交换数据时，BIU 根据 EU 的要求去访问相应内存单元或 I/O 设备，将取出的数据送入指令队列，供 EU 执行指令用。

这两个部件相互作用、互相依赖。但在大多数情况下，各自独立操作。

2．性能及特点

（1）8086/8088 CPU 主要性能

字长：16 位/准 16 位。

时钟频率：8086/8088 标准主频为 5 MHz，8086-2 主频为 8 MHz。

数据、地址总线复用。

内存容量：1 MB。

基本寻址方式：8 种。

指令系统：99 条基本汇编指令，除能完成数据传送、算术运算、逻辑运算、控制转移和处理器控制功能外，还设有硬件支持乘除法指令和串处理指令，可以对位、字节、字、字节串、字串、压缩和非压缩 BCD 码等数据类型进行处理。

端口地址：16 位 I/O 端口地址、可寻址 64K 端口地址。

中断功能：可处理内部软件中断和外部硬件中断，中断源多达 256 个。

支持单片 CPU 或多片 CPU 系统工作。

（2）特点

① 取指令与执行指令重叠并行（指令流水线）

图 2-2　串行顺序执行指令示意

在低档 8 位微机（8080A、Z-80）中，由于取指令和执行指令的衔接关系是串行顺序，如图 2-2 所示。即取第 *K* 条指令操作码，执行第 *K* 条指令；再取第 *K*+1 条指令操作码，执行第 *K*+1 条指令；以此类推。其主要缺点是 CPU 需要等待取指令的时间，致使其执行速度相对较慢。

在 8086/8088 CPU 中，由于 EU 和 BIU 是分开的，取指令和执行指令可以重叠进行，如图 2-3 所示。BIU 指令队列为“先进先出”（FIFO）队列。每当 6/4 个（8086/8088）指令字节中有 2/1 个以上字节空闲，且 EU 也没有要求 BIU 进入总线周期（即不是与外界交换数据的机器周期）时，BIU 就自动执行取指周期，把指令队列填满，以保证 EU 能够连续地执行指令。

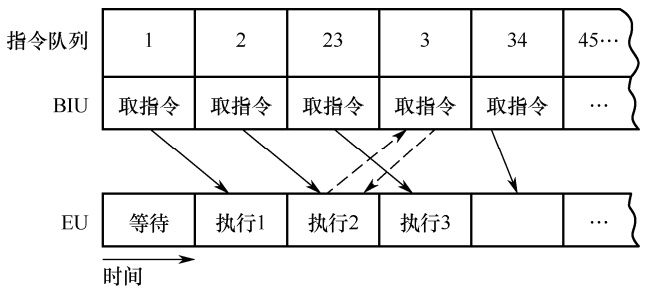


图 2-3　8086/8088 重叠执行指令示意

電子工業出版社

开始时，指令队列是空的，执行部件 EU 处于等待状态。当 BIU 取出第 1 条指令放入指令队列后，EU 控制系统便从队列中取出并由 EU 开始执行第 1 条指令。同时，BIU 取出第 2 条指令，并存入队列中。由于 EU 第 1 条指令尚未执行完，队列未满，于是 BIU 开始取第 3 条指令，这时 EU 才从队列中取出第 2 条指令并执行。在执行第 2 条指令时需要操作数，于是 BIU 从内存中取第 2 条指令的操作数直接送到 EU 中使用。接着，BIU 取出第 4 条、第 5 条指令，而 EU 在执行完第 2 条指令后从指令队列中取出第 3 条指令执行，如此继续。可见，这种指令预取技术把取指令和执行指令的操作重叠进行，这种在现行指令执行时预取下一条指令的技术称为指令流水线（Instruction Pipeline）。由于取消了 CPU 等待取指令时间，因此可加快 CPU 运行速度。

② 段寄存器和存储器分段

8086/8088 有 4 个 16 位段寄存器：代码段寄存器 CS（Code Segment）、数据段寄存器 DS（Data Segment）、堆栈段寄存器 SS(Stack Segment)和附加数据段寄存器 ES（Extra Segment）。

在计算机的内存中存放着如下 3 类信息：

- 代码，即指令操作码，指出 CPU 执行的操作。
- 数据，即数值和字符，程序加工的对象。
- 堆栈，即临时保存的返回地址和中间结果。

为了避免混淆，这三类信息一般分别存放在各自的存储区域内。段寄存器指示这些存储区域的起始地址，或称段基地址。

8086/8088 直接寻址空间为 1 MB，有 20 位地址信息，而它内部只能进行 16 位运算，也就是说，它能处理的地址信息仅 16 位。为解决这一矛盾，把存储器划分为“段”，每个段的物理（实际）长度是 64 KB。4 个段寄存器中，存放表示段起始地址的数据。即 8086/8088 利用段寄存器的内容形成有效地址，对存储器进行访问。

基于存储器的分段结构，在涉及存储器的地址时，必须分清是物理地址还是逻辑地址。物理地址是指 1 MB 存储区域中的某一单元地址，其地址信息是 20 位的二进制代码，以十六进制代码表示是 00000H～FFFFFH 中的一个单元，CPU 访问存储器时，地址总线上送出的是物理地址。编写程序时，则采用逻辑地址，逻辑地址由段地址和偏移量组成。偏移量是在某段内指定存储器单元到段地址的距离。由于访问存储器的操作数类型不同，逻辑地址的来源也不一样。其关系如表 2-1 所示。

表 2-1　访问存储器类型与逻辑地址关系

访问存储器类型	约定段寄存器	可代换段寄存器	偏 移 量	物理地址计算公式
取指令	CS	—	IP	CS×16+IP
堆栈操作	SS	—	SP	SS×16+SP
访问变量	DS	CS，ES，SS	有效地址 EA	DS×16+EA
源字符串	DS	CS，ES，SS	SI	DS×16+SI
目的字符串	ES	—	DI	ES×16+DI
BP 用作基地址寄存器	SS	CS，DS，SS	有效地址 EA	SS×16+EA

物理地址计算公式为

物理（实际）地址=段地址×16+偏移量

物理地址生成示意图如图 2-4 所示。

電子工業出版社

【例 2-1】 设 CS=4232H，IP=66H，则下一条指令地址为

$$\begin{array}{rrl} & 42320\text{H} & \text{代码段地址} \\ + & 66\text{H} & \text{偏移量} \\ \hline & 42386\text{H} & \text{指令物理地址} \end{array}$$

本例中，物理地址与逻辑地址的关系如图 2-5 所示。注意，同一物理地址下，可以有不同的逻辑地址。

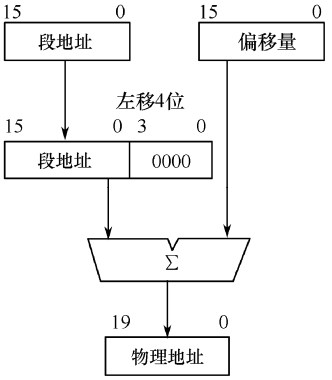


图 2-4 物理地址生成示意图

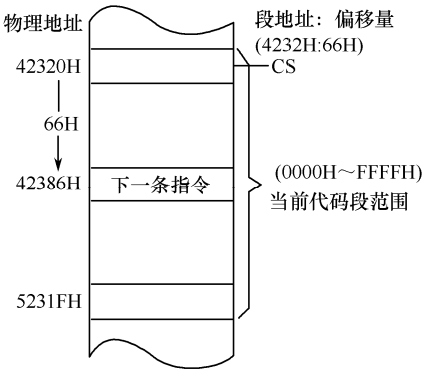


图 2-5 物理地址与逻辑地址的关系

我们可以通过预置段寄存器的内容来访问不同的存储区域。

存储器的分段方式并不是唯一的。存储器各段之间可以连续、错开、部分重叠或完全重叠，这主要取决于各段寄存器的预置内容。对于存储器的一个具体存储单元的物理地址而言，它可以属于一个逻辑段，或同属于几个逻辑段。图 2-6 表示 1 MB 内存储器分成 4 个逻辑段，每个段寄存器分别指示当前的段地址。

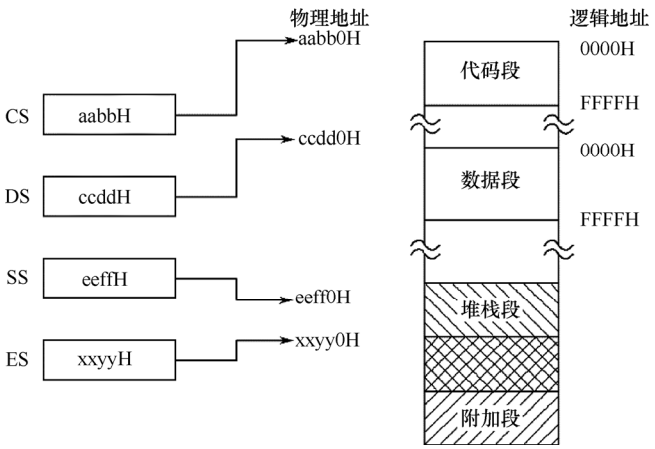


图 2-6 4 个段寄存器分别表示当前 4 个段

③ 部分引脚功能双重定义以适用多处理器

这一点将在 2.3.1 节 8086/8088 引脚中讨论。

3．寄存器配置

8086/8088 CPU 寄存器配置如图 2-7 所示。

電子工業出版社

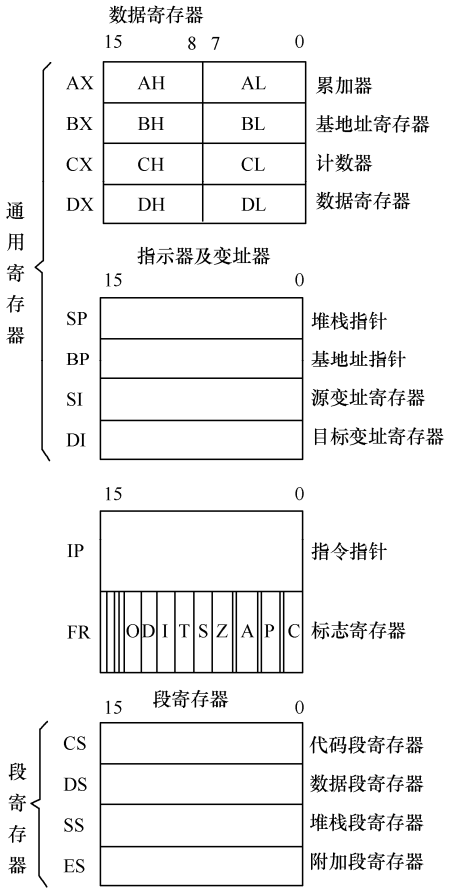


图 2-7　8086/8088 CPU 寄存器配置

（1）通用寄存器

8086/8088 有 8 个 16 位通用寄存器。这 8 个通用寄存器又分为两组。

其中一组称为数据寄存器，包括 AX、BX、CX、DX。它们中的每一个都可分成两个 8 位寄存器：AH、AL；BH、BL；CH、CL；DH、DL。通常，通用寄存器用来存放操作数和中间结果。当处理字节指令时，用 8 位寄存器；当处理字指令时，用 16 位寄存器。

BX、CX、DX 寄存器除了用作算术逻辑运算的操作数外，还有一些特殊用途：① BX 寄存器可在计算地址时用作基地址寄存器；② CX 寄存器在串操作指令中用作计数器；③ DX 寄存器在某些 I/O 操作期间用来保存 I/O 端口地址。

通用寄存器的另一组是指示器和变址器，它们是 4 个 16 位寄存器：堆栈指针 SP（Stack Pointer）、基地址指针 BP（Base Pointer）、源变址寄存器 SI（Source Index）和目的变址寄存器 DI（Destination Index）。

设置这组寄存器目的是：

① 缩短指令代码的长度。

② 在程序运行过程中，允许指令访问这样的存储单元：它们在段内的偏移量正是前面指令的计算结果。在高级语言中，为了建立可变的索引值，常常需要进行这种运算。

③ 用于寄存偏移量，与段寄存器内容相加以获得物理地址。

通常，SP 和 BP 称为指针寄存器，SI 和 DI 称为变址寄存器。指针寄存器为访问现行堆栈段提供了一种方便形式。因此，若不特别指明某个段，则在指针寄存器中存放的偏移量被认为是在现行堆栈段中；类似地，包含在变址寄存器中的偏移量，通常被认为是在现行数据段中。SP 中内容为现行堆栈顶单元在堆栈段中的偏移量，即堆栈单元的位置；而 BP 通常用于存放现行堆栈段的一个数据区“基址”的偏移量。

在串操作时，对变址寄存器 SI 和 DI 的使用也有所区别。被处理的原始数据称为源操作数，其所在单元的偏移量存放于 SI 中；串操作结果的存放地址称为目的地址，其偏移量存放在 DI 中。对串操作来说，SI 和 DI 不能互换。源操作数应位于当前数据段中，目的操作数则应在当前附加段中。

（2）段寄存器

段寄存器在前面已经讨论过，这里不再赘述。

（3）指令指针 IP

IP（Instruction Pointer）指令指针是一个 16 位寄存器，其功能与程序计数器 PC 类似。其内容由 8086/8088 的总线接口部件 BIU 来修改。IP 总是包含下一条指令在当前代码段的偏移量，或 IP 和 CS 一起指出下一条指令的物理地址，即

電子工業出版社

下一条指令的物理地址=CS×16+IP（或 CS×10H+IP）

（4）状态标志寄存器 FR

FR（Flags Register）标志寄存器是 16 位寄存器，但只使用其中的 9 位：CF、PF、AF、ZF、SF、OF 六位用来表示算术和逻辑运算结果特征状态标志；TF、IF 和 DF 三位作为程序控制标志，用来控制 CPU 的工作条件。

FR 标志寄存器各位含义如图 2-8 所示。

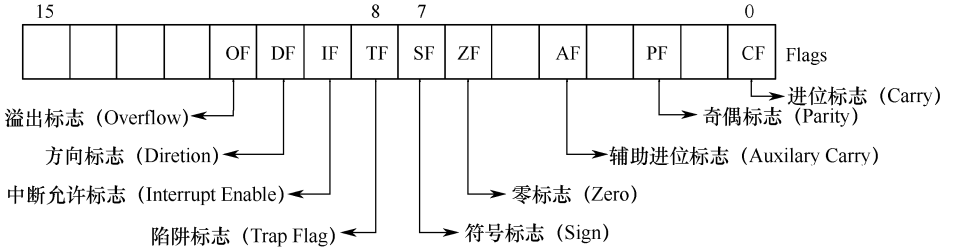


图 2-8　8086/8088 状态标志寄存器

<1> 状态标志

① CF 进位标志。当进行 16 位或 8 位加、减运算时，若最高位（D_{15} 或 D_7）产生进位或借位时，CF=1，否则 CF=0。

② PF 奇偶标志。若运算结果低 8 位中含 1 的个数为偶数时，PF=1；若为奇数，则 PF=0。此标志一般用来检测数据传输中是否发生错误。

③ AF 辅助进位标志。当进行 8 位数或 16 位数运算时，低 4 位向高 4 位（即 D_3 位向 D_4 位）有进位或有借位，则 AF=1，否则 AF=0。此标志用于校正 BCD 编码的十进制算术指令操作。

④ ZF 零标志。若运算结果为 0，ZF=1，否则 ZF=0。

⑤ SF 符号标志。对于带符号的数，用最高位表示数的符号。若运算结果最高位为 1，表示结果为负数，则 SF=1，否则 SF=0。

⑥ OF 溢出标志。当进行带符号数补码运算时，运算结果超出了机器所能表示的数值范围，如字节运算结果大于+127 或小于−128；或字运算时，结果大于+32767 或小于−32768，就产生“溢出”，此时 OF=1，否则 OF=0。在 8086/8088 指令系统中，有一条中断指令 INTO 能够在发生“溢出”时产生内部中断，把程序自动跳转到溢出中断服务子程序。

<2> 控制标志

改变控制标志的状态可以改变处理器的操作。8086/8088 有 3 个控制标志。

① IF 中断允许标志。该标志用于允许或禁止 CPU 响应外部中断，由程序控制。若 IF=1，则 CPU 可以响应外部可屏蔽中断的中断请求；若 IF=0，则禁止 CPU 响应外部可屏蔽中断请求。可以用开中断指令 STI 使 IF 置位和关中断指令 CLI 将 IF 清 0。

② DF 方向标志。用于串操作，由程序控制。若 DF=1，表示执行串操作时，从高地址开始向低地址逐个处理，即串地址自动减量（字节操作减 1，字操作减 2）；若 DF=0，表示从低地址向高地址逐个处理，即串地址自动增量。

③ TF 陷阱标志。该标志是为调试程序而设置的。若 TF=1，表示 CPU 以单步方式执行程序，即 CPU 每执行完一条指令，就自动产生一次内部中断。可以借此来进行程序调试，跟踪执行指令的结果。操作者能逐条执行指令，以检查该指令执行的结果。当 TF=0 时，表

電子工業出版社

示 CPU 正常执行程序。TF 的状态可用指令来设置。

为使读者更好地理解上述状态标志，现举例如下。

【例 2-2】 设有 2345H+3219H，试分析对 FR 的影响。

解：

$$\begin{array}{rl} & 0010\ 0011\ 0100\ 0101 \\ + & 0011\ 0010\ 0001\ 1001 \\ \hline & 0101\ 0101\ 0101\ 1110 \end{array}$$

SF：由于运算结果最高位为 0，所以 SF=0。

ZF：由于运算结果本身不为 0，所以 ZF=0。

AF：由于第 3 位没有向第 4 位进位，所以 AF=0。

PF：由于低 8 位中 1 的个数为奇（5 个 1），所以 PF=0。

CF：由于最高位没有产生进位，所以 CF=0。

OF：由于最高位没有产生进位 CS，次高位没有向最高位进位 CP，所以 OF=0。一般有

$$OF = CS \oplus CP = \begin{cases} 0 & \text{无溢出} \\ 1 & \text{有溢出} \end{cases}$$

注意，OF 表示符号数溢出，而 CF 则表示无符号数溢出（CF=1 溢出，反之则不溢出）。

【例 2-3】 设有 65A0H−B79EH，试分析对 FR 的影响。

$$\begin{array}{rl} & 0110\ 0101\ 1010\ 0000 \\ + & 0100\ 1000\ 0110\ 0010 \quad [B79EH]_{\text{求补}} \\ \hline & 1010\ 1110\ 0000\ 0010 \end{array}$$

结果非零，ZF=0

无进位，有借位 CF=1　　无进位，有借位，AF=1

最高位 1，SF=1

最高位无进位，次高位有进位，OF=1　　低 8 位有 1 个 1，PF=0

用补码加法实现减法时，各位之间产生的进位与对应减法各位产生的借位是相反的。用人工计算过程如下，结果与机器运算完全相同。

$$\begin{array}{rl} & 0110\ 0101\ 1010\ 0000 \\ - & 1011\ 0111\ 1001\ 1110 \\ \hline & 1010\ 1110\ 0000\ 0010 \end{array}$$

结果非零，ZF=0

最高位有借位，CF=1　　有借位，AF=1

最高位 1，SF=1

正数减负数，结果却为负数，　OF=1　　低 8 位有 1 个 1，PF=0

结果表明，若两个操作数是无符号数，CF=1 表示不够减，运算结果是以 2^{16} 为模的差值的补码；若两个操作数是有符号数，则运算结果超出 16 位补码所能表示的范围（OF=1），并不是正确的运算结果。

2.2.2　80386 CPU 内部结构

从 8086 到 80x86/Pentium 微处理器，内部结构虽有不少变化，究其实质仍属 8086 处理

電子工業出版社

器体系，即内部结构各单元均采用并行处理技术，也就是说，微处理器内部多个处理单元可分别进行同步、独立并行操作，以实现高效流水线工作，避免串行处理，最大限度地发挥了处理器性能。本节至 2.2.4 节将重点说明 80386、Pentium CPU 的结构特点，并简要介绍数学协处理器 80x87。

8086/8088 后是 16 位数据线、24 位地址线的 80286 CPU，其内部结构由 4 部分组成：执行单元 EU、地址单元 AU、总线单元 BU 和指令单元 IU，采用流水线工作方式，并行操作，速度比 8086 CPU 快 5 倍。80286 CPU 支持两种工作模式，即实地址实模式和虚地址保护模式。当 80286 CPU 工作在保护模式时，能够支持多任务，处理器提供虚拟内存管理和多任务的硬件控制。在保护方式下 80286 CPU 可产生 24 位物理地址，使用 16MB 内存，并产生 1024MB（1GB）虚拟内存。

80386 CPU 内部、外部的数据总线、地址总线均为 32 位，直接寻址能力达 4 GB，虚地址保护模式下的存储空间可达 64 TB，图 2-9 为 80386 CPU 内部结构。

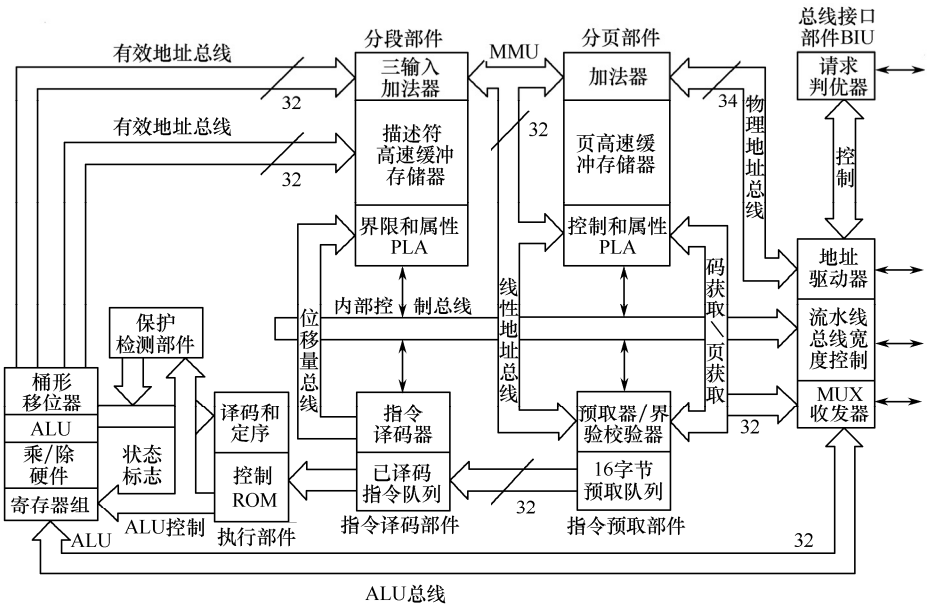


图 2-9　80386 CPU 内部结构

80386 CPU 结构比 8086/8088、80286 复杂，有较多的并行处理单元，增强了 80386 CPU 的性能。依其职能共有 6 个处理单元：执行单元、分段单元、分页单元、总线单元、指令预取单元和译码单元。下面简要介绍各处理单元的功能。

（1）总线单元

总线单元是 80386 CPU 通往外部世界的接口。总线接口提供 32 位数据总线、32 位地址总线以及控制总线传输的一些信号，采取多路分解方法实现 8 位、16 位和 32 位数据传输。地址和数据的多路分解不但改进了性能，而且简化了硬件设计。

总线单元负责 CPU 内部各部件与存储器、输入/输出接口之间数据或指令的传输。CPU 内部的其他部件都能与总线单元直接通信，并将它们的总线请求传输给总线单元。在指令执行的不同阶段，指令、操作数以及存储器偏移地址都可从存储器取出送到 CPU 内部的有关部件。但当 CPU 内部多个部件同时请求使用总线时，为了使程序的执行不被延误，总线

电子工业出版社

单元的请求优先控制器将优先响应数据的传输请求，只有不执行数据传输操作时，总线单元才可以满足预取指令的请求。从图 2-9 可以看到，在数据访问时，存储地址来源于分页单元；在代码访问时，存储地址由指令预提取单元提供。

（2）指令预取单元

80386 指令代码的预取不再由 BIU 负责，而由一个独立的指令代码预取单元完成。指令预取单元由预取器和预取队列组成。预取器管理一个预取指令指针和段预取界限，从分段单元来的线性地址和分段界限分别送到这里。当总线单元不执行指令执行部分的总线周期时，若预取队列存在空单元或发生一次控制转移，预取器便通过分页单元向总线单元发出指令预取请求。分页单元将预取指令指针送出的线性地址变为物理地址，再由总线单元及系统总线从内存单元中预取出指令代码，放入预取队列中。80386 CPU 的预取队列可存放 16 字节的指令代码。进入预取队列的指令代码将被送至指令译码单元进行译码。预取器保持预取队列总是满的。

假如指令预取单元中队列是满的，而且执行单元也不要求存储器提供操作数，则总线接口单元不需要完成总线周期，处于空闲状态。

（3）指令译码单元

从图 2-9 可以看到，译码单元接受指令预取单元的指令队列的输出，读入的形式是机器代码指令，将其译码为微代码指令形式供执行单元使用。这些内部编码中包含了控制其他处理部件的各种控制信号，因而指令译码单元为指令的执行做好了准备。在 80386 CPU 的指令单元内，指令队列允许有三条指令被译码以供执行单元使用，进一步提升了 CPU 的性能。

（4）执行单元

执行单元由控制部件、数据处理部件和保护检测部件组成，其任务是将已译码指令队列中的内部编码变成按时间顺序排列的一系列控制信息，并发送给 CPU 的其他处理部件，以便完成一条指令的执行。具体地说，执行单元内包含算术/逻辑单元（ALU）、通用寄存器、专用乘法/除法器和移位器，还有控制 ROM（只读存储器）。控制 ROM 内包含微代码序列，它们是完成 80386 CPU 机器代码指令的微程序。执行单元从指令队列中读出已译码的指令并实现其操作。ALU 实现指令所要求的算术、逻辑和移位操作。当指令执行时，如果需要，段和页单元产生操作数据的地址并由总线接口单元完成读或写周期，从存储器或 I/O 设备中存取操作数。此外，ALU 采用特定硬件模块来完成乘法、除法、移位和循环移位等操作，这使指令完成上述操作时的性能得到大大改善。

执行部件中还设有一条附加的 32 位内部总线和专门的控制逻辑部件，并提供同时执行两条指令所需要的控制回路，可使每条访问存储器的指令与前一条指令并行执行。

（5）分段、分页单元

分段、分页单元为 80386 提供存储器管理和保护服务，负责地址产生、地址转换和对总线接口单元的段检查，因此进一步提高了 CPU 性能。80386 CPU 存储器管理的分段模型是用硬件完成高速地址计算、从逻辑到线性地址的转换和保护性检查。例如，在实模式下，执行单元要求分段单元将代码段（CS）寄存器的内容加上指令指针（IP）的数值，得到下一个要提取的指令地址，将 20 位的物理地址输出到地址总线，最后到达总线单元。

在保护模式下，分段单元完成逻辑地址到线性地址的转换，并在总线周期内完成多种保护性检查。分页单元实现 80386 存储器管理的分页模型，具有转换后备缓冲器，用于存放近期使用的页目录和页表。当分页工作被允许时，分段单元产生的线性地址被分页单元当作

电子工业出版社

输入。分页单元把线性地址转换成存储器和 I/O 能接受的物理地址，并输入到总线接口单元。

80386 CPU 有 7 大类 32 个寄存器：通用寄存器、段寄存器、指令指针和标志寄存器、控制寄存器、系统地址寄存器、调试寄存器、测试寄存器，如图 2-10 所示。

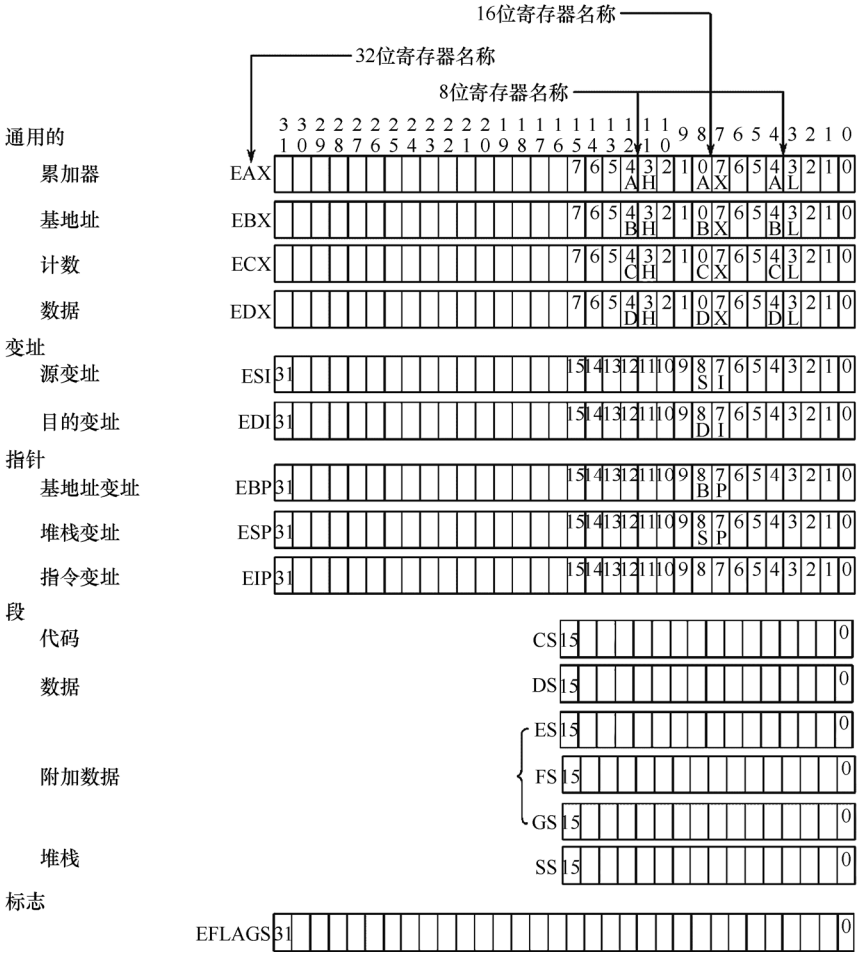


图 2-10　80386 CPU 的基本寄存器集

（1）通用寄存器

80386 的 8 个 32 位通用寄存器 EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI 都是由 8086/8088 的相应 16 位寄存器 AX、BX、CX、DX、SI、DI、BP、SP 扩展而来。32 位通用寄存器的低 16 位可单独使用，而 AX、BX、CX、DX 的高、低 8 位可分别作为 8 位寄存器使用。

（2）段寄存器

80386 的段寄存器仍是 16 位，共有 6 个：CS、DS、ES、FS、GS、SS。其中 CS、DS、SS、ES 与 8086/8088 相同，而 FS、GS 是 80386 扩充的两个附加数据段寄存器。

在实模式下，80386 与 8086/8088 段寄存器的使用完全相同。

在保护模式下，存储单元的逻辑地址仍由两部分组成，即段基址和段内偏移地址。段内偏移地址为 32 位，段基址也是 32 位，但并不由段寄存器的值直接确定，而是以段寄存器（段选择符）为索引从描述表中找到所指向的描述符，再从描述符中找到段基址后与段

电子工业出版社

内偏移地址一起确定物理地址。下面将分别介绍段描述符和段选择符。

段描述符的格式如图 2-11 所示。低 48 位是 80286 描述符，包括 16 位段界限（也称为段边界）、24 位段基地址和 8 位属性。80386 以上的 32 位 CPU 则在此基础上扩充了 8 位基地址、4 位界限和 4 位属性，即为 20 位段界限、32 位段基地址和 12 位属性。

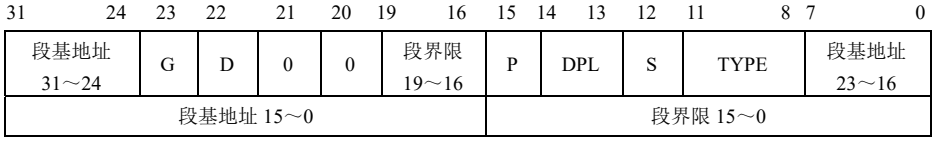


图 2-11　段描述符格式

段基地址为段起始地址，段界限给出段内最大偏移地址，其他位均为属性，具体含义如下。

① G（Granularity）：粒度值（段界限所用单位）。G＝0 表示段界限给出的是段内最大偏移地址，以字节为单位，段界限最大为 FFFFFH，即段的最大长度为 1MB；G＝1 表示单位是页，每页大小为 4KB，段的最大长度为 4GB。

② D：对于代码段、可扩展数据段和堆栈段，其含义有所不同。

- 可执行代码段，表示指令中默认操作数的长度。D=1 表示操作数长度为 32 位或 8 位，D=0 表示操作数长度为 16 位或 8 位。
- 可扩展数据段，表示段的上边界。D=1 表示 4GB；D=0 表示 64KB。
- 堆栈段，表示堆栈操作指令（PUSH 和 POP）隐含使用的寄存器是 ESP（D=1）还是 SP（D=0）。

③ P（Present）：存在位。P=1 表示存在，即在实际内存中。

④ DPL（Descriptor Privilege Level）：描述符特权级（2 位），可以为 0（00）～3（11）级，0 级为最高级。

⑤ S（Segment）：段描述符类别。S=1 表示代码段或数据段描述符；S=0 表示系统描述符。

⑥ TYPE：段的类型（4 位）。说明访问、读/写控制及扩展特点等，其中类型值的最高位（可写为 E）表明描述的是数据段还是代码段。E=0 表明段是数据段，相应描述符为数据段（包括堆栈段）描述符，数据段是不可执行的，但总是可读的；E=1 表明是可执行段，即代码段，其总是不可写的。类型值的最低位（可写为 A）表明该段是否被访问过。A=0 表示尚未被访问；A=1 表示已访问过。段描述符具体类型说明如表 2-2 所示。

表 2-2　段描述符类型说明

类 型 值	含　义	类 型 值	含　义
0000	只读	1000	只执行
0001	只读，已访问	1001	只执行，已访问
0010	读/写	1010	执行/读
0011	读/写，已访问	1011	执行/读，已访问
0100	只读，向下扩展	1100	只执行，一致代码段
0101	只读，向下扩展，已访问	1101	只执行，一致代码段，已访问
0110	读/写，向下扩展	1110	执行/读，一致代码段
0111	读/写，向下扩展，已访问	1111	执行/读，一致代码段，已访问

例如，某可读/写的有效（存在的）数据段的描述符为 0000 F210 0000 FFFFH，对照图 2-11 格式，第 3 字节为 F2H，即 P=1，DPL=3，S=1，TYPE＝2，是特权级为 3 的可读/

电子工业出版社

写数据段；第 2 字节为 00H，即 G=0，D=0，表明该段为 16 位段，其段界限单位为字节，由所有段基地址字段可得其基地址为 100000H，段界限是 0FFFFH。

再如，某可执行的 32 位代码段的描述符为 12C0 9834 5678 0010H，同样对照图 2-11 格式，可得其基地址是 12345678H，以 4 KB 为单位的段界限值是 10H（G=1），描述符特权级 DPL=0（最高特权级）。

段描述符存放在两个系统表 GDT 和 LDT 中。GDT 是全局描述符表，用于给出系统使用的和各任务共用的段描述符，换言之，其可用来管理全局存储器地址空间，GDT 只有一个；LDT 是局部描述符表，存放某个任务专用的段描述符，用于管理任务用到的局部存储器地址空间，LDT 表可以有若干。

段寄存器中存放的是段描述符的 16 位选择符，其格式如图 2-12 所示。其中 b_1b_0 位为请求特权级 RPL（Requestor Privilege Level）字段，与 DPL 类似，为 0～3 级（2 位），在数据访问的特权级检查中将使用该字段。b_2 位为表指示符字段 TI，指明本描述符是在 GDT（TI=0）还是在 LDT（TI=1）中，b_{15}～b_3 共 13 位构成描述符索引字段 INDEX，用于指明段描述符在指定的描述符表中的序号。

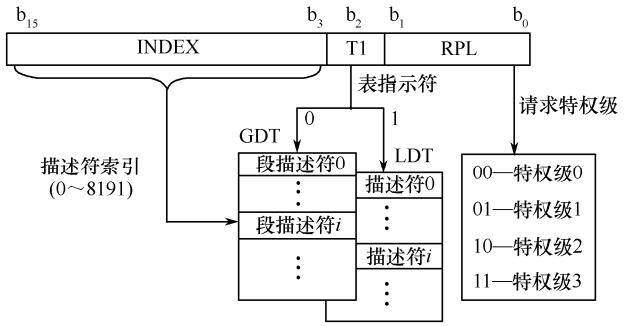


图 2-12　段选择符格式

当将一个选择符装入一个段寄存器时，处理器将自动从 GDT 或 LDT 中找到其对应的描述符并装入相应段寄存器的描述符高速缓存器中，该过程对用户是透明的。以后，每当访问存储器时，与所用段相关的段描述符高速缓存器就自动参与存储器访问操作。使用描述符高速缓存器的原因是，在同一段访问时，可直接使用该寄存器内的描述符信息，而无须重新从 GDT 或 LDT 表中去取，大大加快了访问时间。

在高速缓存器中获取的段基地址成为线性地址或物理地址计算中的一个分量，界限用于段限检查，属性则对照所要求的存储器访问类型进行检查，即

线性地址=段描述符高速缓存器中的段基址+偏移地址

不使用页部件时，线性地址即为物理地址；使用页部件时，线性地址需经页管理部件使用页目录表和页表转换成物理地址。

（3）指令指针和标志寄存器

80386 的指令指针寄存器 EIP 也是由 8086/8088 的 16 位指令指针寄存器 IP 扩展而得到的。EIP 中存放的是下一条要执行指令在代码段的偏移地址。80386 在实方式下采用 16 位的指令指针 IP。80386 的标志寄存器是 32 位，如图 2-13 所示。其中低 16 位中比 8086/8088 增加了 IOPL（第 12、13 位）和 NT（第 14 位），高 16 位中只使用了 2 个标志位：RF（第 16 位）和 VM（第 17 位）。

電子工業出版社

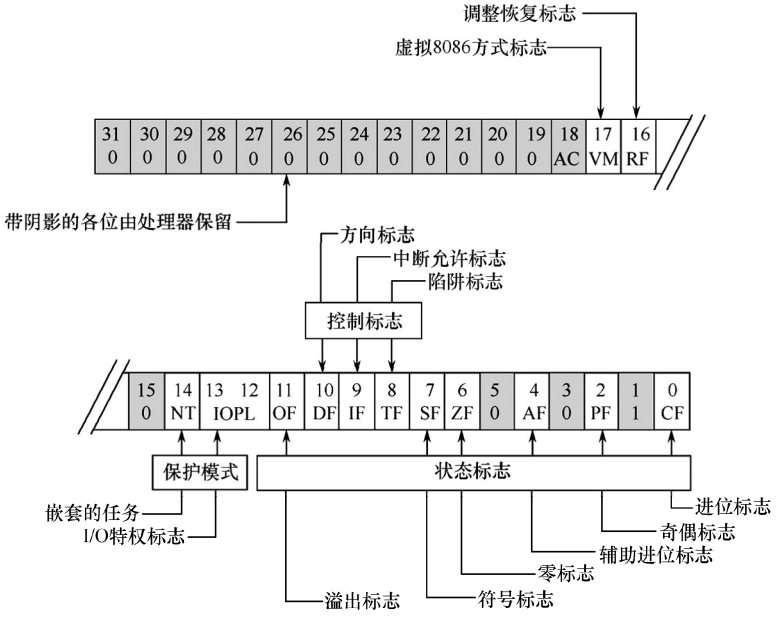


图 2-13　80386 CPU 的标志寄存器

这些新增标志位的含义如下。

① IOPL（I/O Priority Level）：I/O 特权标志位，只适用保护模式，指明 I/O 操作的级别。

② NT（Nested Task）：嵌套标志位。当前执行的任务正嵌套在另一任务中时，NT=1；否则，NT=0。NT 也仅适用保护模式。

③ VM（Virtual 8086 Mode）：虚拟 8086 模式标志位。处于保护模式的 80386 转为虚拟 8086 方式时 VM=1。

④ RF（Resume Flag）：恢复标志位，配合调试寄存器的断点或单步操作一起使用。在处理断点时，在两条指令之间检查到 RF=1 时，则下一条指令执行时的调试故障被忽略。在成功地完成一条指令后，RF=0；而当接收到一个非调试故障的故障信号时，RF=1。

（4）控制寄存器

80386 有 4 个 32 位的控制寄存器 CR_0、CR_1、CR_2 和 CR_3，如图 2-14 所示，它们的作用是保存全局性机器状态。

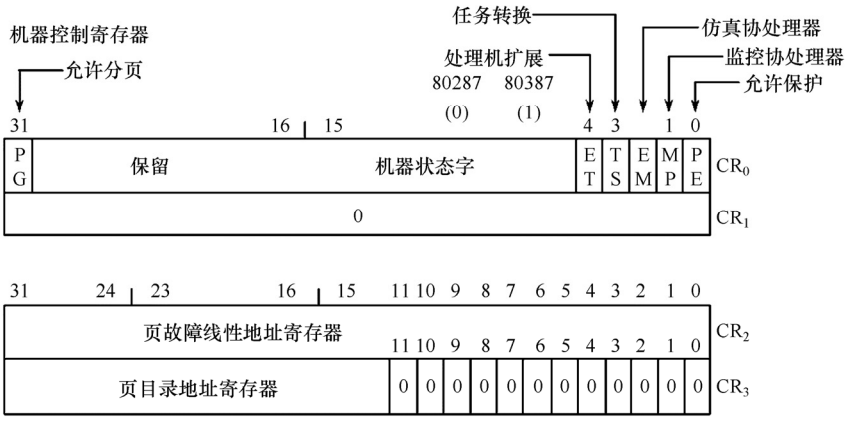


图 2-14　80386 CPU 的控制寄存器

電子工業出版社

CR_0 定义了如下 6 位。

① PE（Protection Enable）：允许保护标志，用于控制微处理器是否进入保护模式。若 PE=1，则 80386 转换成保护模式。PE 复位后，微处理器回到实地址模式。系统加电后，微处理器总是初始化为实地址模式。

② MP（Monitor Processor extension flag）：协处理器监控标志，监控数学协处理器是否存在。若系统中有一个数学协处理器存在，则 MP=1；否则，MP=0。

③ EM（EMulate processor extension flag）：仿真协处理器标志，表示是否用软件仿真数学协处理器。若 EM=0，所有协处理器操作在实际协处理器 80387 上执行；若 EM=1，将采用软件仿真实现协处理器的功能。

④ TS（Task Switched）：任务切换标志，由硬件置位而由软件复位。当一个任务转换完成后，TS 标志自动置 1。

⑤ ET（Extension Type）：扩展类型标志，表示系统中所使用的数学协处理器的类型。该标志设置的原因在于 80386DX 可以支持 80287 和 80387。若 ET=1，表示系统中使用的数学协处理器是 80387；若 ET=0，表明使用的是 80287 或没有使用数学协处理器。

⑥ PG（PaGing flag）：允许分页标志，表示处理器是否允许对存储器线性地址进行分页，并进行物理地址转换。若 PG=1，允许分页，并由分页单元将线性地址转换为物理地址；若 PG=0，禁止分页，线性地址直接作为物理地址使用。

CR_1：未定义的控制寄存器，供后期的微处理器使用。

CR_2：页故障线性地址寄存器，保存最后出现页故障的全 32 位线性地址。

CR_3：页目录基地址寄存器，保存页目录表的物理基地址。

（5）系统地址寄存器

80386 CPU 设置了 4 个系统地址寄存器，保存操作系统所需要的保护信息和地址转换表信息，如图 2-15 所示。

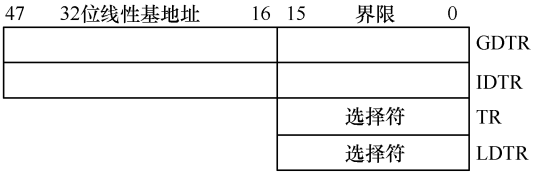


图 2-15　80386 CPU 的系统地址寄存器

① GDTR（Global Descriptor Table Register）：全局描述符表寄存器，48 位寄存器，保存全局描述符表的 32 位线性地址和 16 位段界限。

② IDTR（Interrupt Descriptor Table Register）：中断描述符表寄存器，48 位寄存器，保存中断描述符表的 32 位线性地址和 16 位段界限。

③ LDTR（Local Descriptor Table Register）：局部描述符表寄存器，16 位寄存器，保存当前局部描述符表的 16 位选择符。

④ TR（Task state Register）：任务状态寄存器，16 位寄存器，保存当前任务的任务状态段的 16 位选择符。

前文已述，段描述符存放在全局描述符表 GDT 或局部描述符表 LDT 中。事实上，80386 有 3 种描述符表，除了全局和局部描述符表，还有中断描述符表 IDT（Interrupt Descriptor Table），给出它们存放位置的寄存器分别是 GDTR、LDTR 和 IDTR。

電子工業出版社

如图 2-16 所示，GDTR 是 48 位寄存器。该寄存器最低 2 字节为限长（Limit）字节，它规定了 GDT 按字节计算的大小。由于限长为 16 位，GDT 最大可达 65535 字节。GDTR 的高 4 字节为基址（Base），指示物理存储器中 GDT 的起始位置。该 32 位基址允许该表定位在 80386 线性地址空间的任何地方。

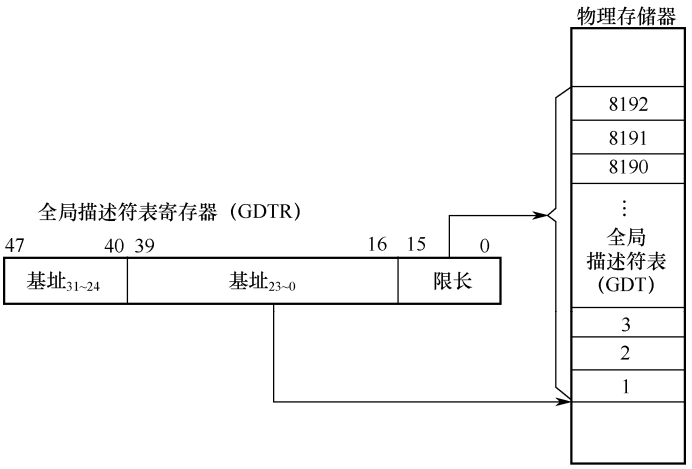


图 2-16　全局描述符机制

同全局描述符表寄存器一样，中断描述符表寄存器 IDTR 在物理存储器中给出了中断描述符表 IDT 的所在位置，其工作机制与 GDTR 相似。IDTR 和 IDT 描述符表提供一种机制，将程序控制转给中断并执行中断服务程序。

局部描述符表寄存器 LDTR 用于定位当前局部描述符表 LDT。“当前”是因为在保护模式下，系统支持多任务，每个任务除了可访问全局描述符表外还可访问它自己的描述符表。

局部描述符表 LDT 定义了任务用到的局部存储器地址空间。由于每项任务都有它自己的存储器段，因此保护模式的软件系统可能会包含多个局部描述符表。图 2-17(a)说明了 LDT_0 到 LDT_n 的关系。

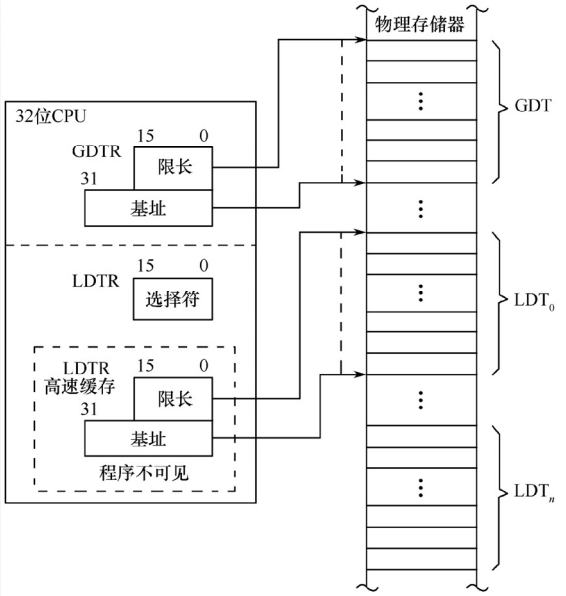
图 2-17(b)说明 16 位的 LDTR 值并不直接定义一个局部描述符表，只是一个指向 GDT 中 LDT 描述符的选择符。如果 LDTR 中装入了选择符，相应的描述符就能够从全局存储器中读出来并装入局部描述符表高速缓存中，只有该描述符才能确定局部描述符表。32 位的“基址”值标识物理存储器中表的起始地址，16 位的“限长”值定义表的大小。将该描述符装入高速缓存就为当前任务创建了一个 LDT。也就是说，每次 LDTR 中装入选择符，则局部描述符表的描述符被缓存，激活一个新的 LDT。

在三种描述符表中存放的不仅是段描述符，还有另一种描述符——系统描述符，它的一般格式如图 2-18 所示。

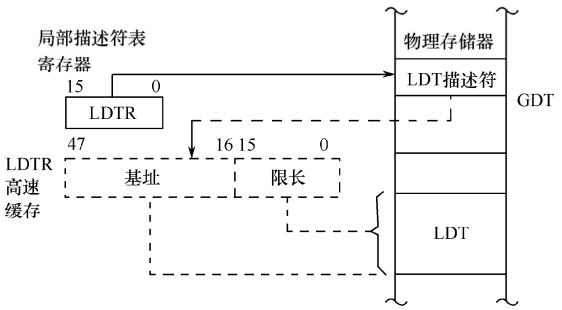
显然，系统描述符和图 2-11 中的段描述符类似，其中 G、P、DPL 的含义均相同，但 TYPE 类型值的意义不同，80386 系统中有 16 种可能的系统描述符类型（如表 2-3 所示），其中有些类型是为 80286 定义的，有些则还没有定义，预留给后续处理器。

系统描述符主要分为门描述符（“门”是保护模式下实现程序控制转移的方式）和任务状态段 TSS（在多任务切换时用于保存当前任务状态）描述符，其中门描述符又分为调用门、任务门、中断门、陷阱门。

電子工業出版社



(a) 多个局部描述符表的示例



(b) 局部描述符表寄存器LDE的工作机制

图 2-17　局部描述符机制

31 24	23	22	21	20	19 16	15	14 13	12	11 8	7 0
段基地址 31～24	G	0	0	0	段界限 19～16	P	DPL	0	TYPE	段基地址 23～16
段基地址 15～0						段界限 15～0				

图 2-18　系统描述符格式

表 2-3　80386 系统描述符类型

类　型	用　途	类　型	用　途
0000	无效	1000	无效
0001	可用的 80286 TSS	1001	可用的 80386 TSS
0010	LDT	1010	保留
0011	正在执行的 80286 TSS	1011	正在执行的 80386 TSS
0100	80286 调用门	1100	80386 调用门
0101	任务门	1101	保留
0110	80286 中断门	1110	80386 中断门
0111	80286 陷阱门	1111	80386 陷阱门

電子工業出版社

任务状态段描述符就是将图 2-18 中的系统描述符中类型值按表 2-3 对应使用，而门描述符是一种特殊格式的系统描述符，其具体格式如图 2-19 所示。门描述符包括一个 32 位的偏移地址、一个字计数器和一个选择符，表示属性的 P 和 DPL 的意义与其他描述符意义相同。

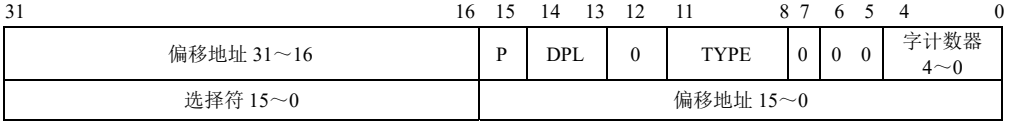


图 2-19　门描述符格式

32 位的偏移地址和选择符用于给出转移程序的地址。字计数器仅对调用门有意义，在使用调用门的过程中，通常需要使用堆栈来传递一些参数，这个字计数器用来计数传递的双字数据的个数。

任务寄存器 TR 存放的是当前任务状态段 TSS 的选择符，它在保护模式任务切换机制中很重要。任务切换的实质是用一个新任务的 TSS 段选择符加载任务寄存器 TR。TR 的初值由软件（通常是操作系统）装入，之后将开始执行第一个任务，其后，执行任务切换时相应地修改选择符。

如图 2-20 所示，任务寄存器 TR 中的选择符指示全局描述符表中 TSS 描述符的位置。注意当选择符装入 TR 时，相应的任务状态段 TSS 描述符自动从存储器中读出并装入任务描述符缓存器中，其工作特性与前述段描述符的选择符特性相同，但特别指出的是，TSS 描述符只能保存在 GDT 中，而不能在 LDT 和 IDT 中。

（6）调试寄存器

如图 2-21 所示，80386 CPU 中设置了 8 个调试寄存器 DR_0～DR_7，它们为程序调试提供了硬件支持。其中，DR_0～DR_3 为线性断点寄存器，可保存 4 个断点地址，从而方便程序设计人员按照调试意图组合指令的执行。DR_4 和 DR_5 保留。DR_6 用于保存断点状态，DR_7 用于控制断点的设置。

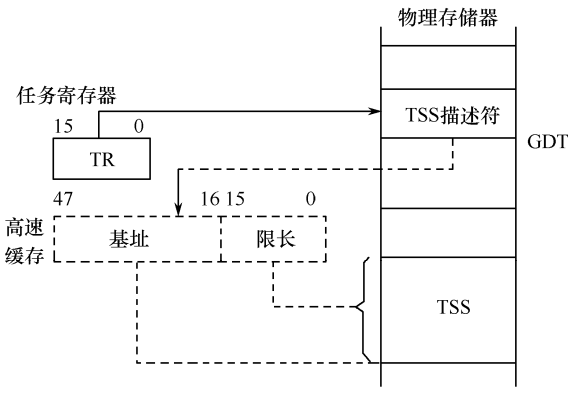


图 2-20　任务寄存器 TR 及其工作特性

图 2-21　调试寄存器

（7）测试寄存器

80386 CPU 中设置了 2 个 32 位的测试寄存器 TR_6 和 TR_7，TR_6 为测试命令寄存器，用于对 RAM 和相关寄存器进行测试；TR_7 用于保留测试后的结果。

80386 CPU 提供 3 种工作模式：实地址模式、虚地址保护模式、虚拟 8086 模式（本质

电子工业出版社

上也属于保护模式）。其地址种类也有 3 种：逻辑地址、线性地址和物理地址。虚地址保护模式提供两种主要的保护功能：一是通过给每个任务分配不同的虚拟地址空间，使任务之间完全隔离，每个任务有不同的虚拟地址到物理地址的转换映射；二是任务内的保护机制操作，以保护系统存储段及特别的处理器寄存器，使其不被其他应用程序所破坏。为了实现这些功能，CPU 以 4 个特权级（0～3 级，特权级依次降低）来隔离或保护各用户及系统，每个存储段都与同一个特权级相联系，只有足够级别的程序段才可对相应的段进行访问。

虚地址保护模式下的存储器组织相当复杂。这里仅考虑存储器管理单元（MMU）是如何管理地址空间，以及如何将虚拟（逻辑）地址转换成物理地址，不讨论保护功能。

保护模式的存储器管理单元使用 48 位的存储器指针。指针分为两部分：选择符和偏移量。该 48 位存储器指针称为虚拟地址，在程序中用以规定指令或数据的存储位置。选择符长 16 位（见图 2-12），偏移量长 32 位。其中选择符可放在 CPU 的段选择符寄存器中，比如要访问存储器中的指令代码，则活动段选择符一定放在 CS 中。指针的这一部分实际上选择的是 CPU 虚拟地址空间中一个特定的段。偏移量放在 CPU 用户可访问的寄存器中，或用于直接寻址。如进行指令代码的访问（取指），偏移量可放在 EIP 寄存器中，指针提供要访问的存储器段中的偏移量，即要执行的指令码的第 1 字节的相对偏移地址。由于偏移量长 32 位，段空间可达 4 GB。

由图 2-12 可知，16 位选择符又分为 13 位索引、1 位表选择位和 2 位请求特权级。其中 2 位 RPL 并不用于存储器段选择，因此 16 位中只有 14 位用于寻址，这样虚拟地址空间可容纳 2^{14}（16384=16K）个存储器段，每个段最大可达 4 GB。这些段就是 CPU 的 MMU 管理虚拟地址空间的基本元素。

另一种查看虚拟地址空间大小的办法是将 14 位段选择符和 32 位偏移量结合起来，得到 46 位虚拟地址，从而虚拟地址空间可包含 2^{46} B=64 TB。

80386 的存储器管理单元可以实现虚存的分段和分页机制。在分段机制中，64 TB 虚拟地址空间分成了 32 TB 的全局存储器地址空间和 32 TB 的局部存储器地址空间。其划分大致如图 2-22 所示，由选择符的 TI 位选择是全局还是局部描述符表，每个地址空间可允许最多 8192 个存储器段。这种情况假定了全局描述符表和局部描述符表中的每个描述符都在使用中且设为最大值，实际情况下段的总数和各段大小可选定。描述符定义了相应段的属性，但是在实际系统应用中并不是所有的描述符都在使用。

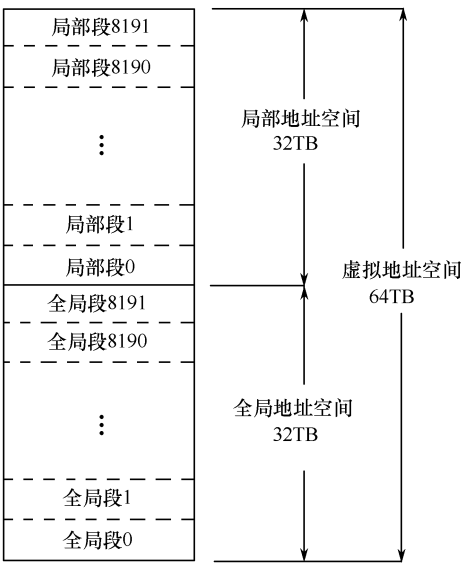


图 2-22　虚拟地址空间的划分

80386 支持分段和分页两种存储管理机制，可以将 48 位虚拟地址映射到硬件所需的 32 位物理地址。

分段机制下各个存储块大小不一，可以在 4 GB 范围内自行定义，每个段最多可以分配 4 GB 的物理地址空间，最少 1B 。物理空间多次交换后可能变得非常零碎，因此存储和管理的对象较复杂。

分页机制下的对象是固定大小的存储块，如 4 KB（80386 及以上）或 4 MB（Pentium 及以上）等，称为页（Page）。这时整个线性地址空间和整个物理地址空间都可看成由很多页组成，线性地址空间的任何一页都可以映射到物理地址空间的

電子工業出版社

任何一页。

80386 CPU 地址转换过程如图 2-23(a)所示。其过程是，先对虚拟（逻辑）地址进行段转换，分段机制可由产生的线性地址直接作为物理地址进行处理，这时如禁止分页机制（PG=0），发生的地址转换过程如图 2-23(b)中阴影部分所示。如果允许采用分页机制（PG=1），线性地址再经过分页转换产生物理地址。

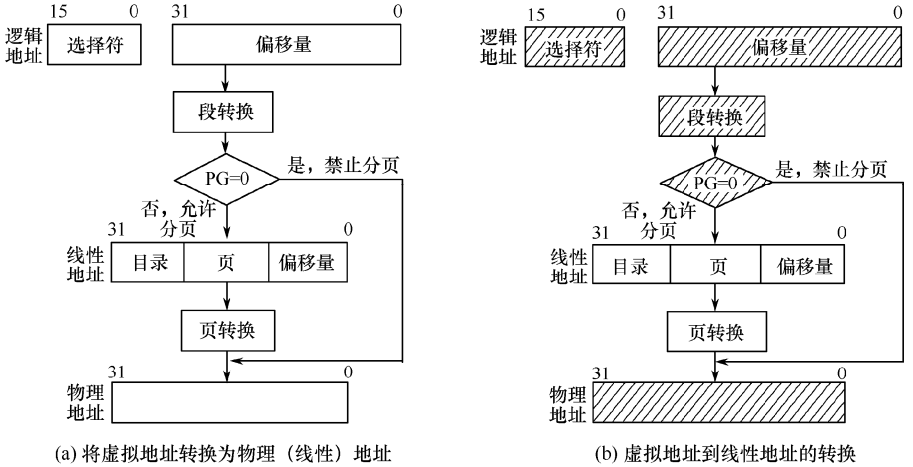


图 2-23　80386 CPU 地址转换过程

作为地址转换的一部分，MMU 还需确定虚拟地址空间的相应段或页是否在物理存储器中，如果存储器中已有该段或页，操作正常；如果不在，将此情况视为异常（Exception）。该异常将激活存储管理程序从外部存储器向物理存储器（内存）装入该段或页，这一过程称为交换，即旧的段或页换到磁盘上，新的段或页则换入刚腾出空间的物理存储器中。尽管发生了交换，但在程序看来所有的段或页都在存储器中。80386 的保护模式支持存储器地址空间的分页组织，其分页机制提供了虚拟存储管理大小合适的存储块，避免了分段机制管理的复杂性。若 80386 CPU 采用分页组织，则物理地址空间被划分成 1048496 页，每页长 4096 B（4 KB）。图 2-24 表示物理地址空间按这种方式的组织形式。分页存储器使用这种固定大小块有一个缺点，存储管理程序每次都最少分配 4 KB，即使它们并不全用，降低了空间利用率，但分页机制避免了大小不一的分段反复交换造成的存储区碎片，简化了存储管理程序。

从图 2-23(a)可看到，段转换过程产生的线性地址不再直接用作物理地址，还要经过称为页转换的第二次转换过程。图 2-25 是页存储机制下的 32 位线性地址格式，由三部分（域）组成，形成两级分页机制。

图 2-24　80386 物理地址空间的分页管理

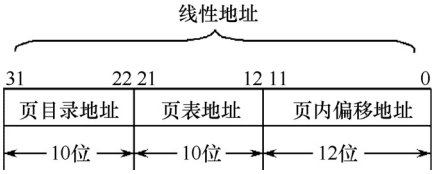


图 2-25　页存储机制下的线性地址格式

電子工業出版社

① 页目录地址，第 31～22 位（10 位）。说明页表在页目录表中的位置，即应选择第几号页表。该值乘以 4 即为页表基地址在页目录表中的相对地址。页目录表基地址由 CR_3 中页目录基址寄存器（PDBR）确定。

② 页表地址，第 21～12 位（10 位）。说明页在页表中的位置，即应选择第几号页。该值乘以 4 即为页基地址在页表中的相对地址。页目录地址和页表地址确定了物理地址的高 20 位（A_{31}～A_{12}）。

③ 页内偏移地址，第 11～0 位（12 位）。说明某寻址单元在页中的相对位置。

80386 在分页方式下，以页（4 KB）为单位管理内存。对于某一任务，可以将用到的存储页的基地址（32 位）存放到一个或几个页表中。每个页表的长度为 4 KB，可以存放 1024 个页的基地址。所有页表的基地址（32 位）存放到一个页目录表中，每个页目录表可以存放 1024 个页表的基地址。页目录表的基地址则由指令装载到控制寄存器 CR_3 中。

图 2-26 表示两级分页机制下的线性地址是怎样转换成对应的物理地址的。存储器页目录表地址由 CR_3 中页目录基址寄存器（PDBR）确定。这 20 位实际上决定该目录基址的高 20 位。低 12 位为相对页首地址（低 12 位为 0）的偏移量。分页机制避免了大小不一分段交换造成的存储区碎片。

实际上，线性地址中 10 位目录域是相对于 PDBR 的偏移量，由此获得 32 位页目录项指针，并缓存在转换检测缓冲器（TLB）中，其值被用作存储器中页表的基址。再结合线性地址的 10 位页域得到页帧表基址，并也被缓存在转换检测缓冲器中。由图 2-26 可知，由此基址加上线性地址 12 位偏移量则可标识出操作数在活动页帧中的位置。

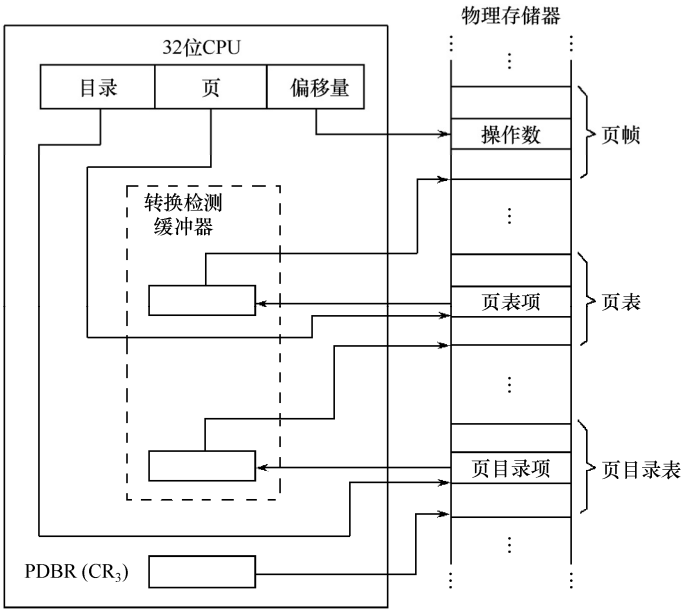


图 2-26　分页机制下将线性地址转换成物理地址

80386 转换检测缓冲器实际可以保存 32 个表项，因此有 128 KB 存储区可直接被访问。这部分存储器中的操作数不需再从页表中读出而可直接存取。如果要访问的操作数不在这些页中，则产生将页表项读进转换检测缓冲器的额外开销。

80386 虚拟地址分页转换中涉及了页表项，只有两种类型的页表项：页目录项和页表项，

電子工業出版社

接下来将给出目录项和页表项的结构。

单一页目录或页表项的格式如图 2-27 所示。如果表项在页目录表中，则最左边的 20 位是页表的基址；如果表项在页表中，则这 20 位表示页帧的基址。注意到表项只提供基址的第 12～31 位，最右边的 12 位假定为 0，这样页表和页帧总是位于 4 KB 地址边界上。从图 2-26 中看到，该表项被缓存到转换检测缓冲器中。

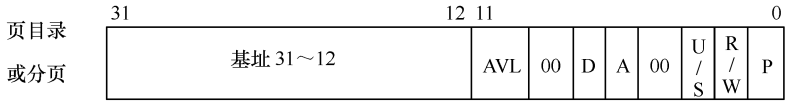


图 2-27　目录项或页表项格式

表项的低 12 位提供页表或页帧使用的保护属性或统计信息。例如，用户/管理员（U/S）和读/写（R/W）位实施两级页保护机制。将 U/S 位设为 1，则选择用户级保护，用户级是较低的特权级，相当于分段模型中的保护级 3。也就是说，用户级指定给那些可被应用程序访问的存储器页。另一方面，将 U/S 设为 0，则指定对表或帧进行管理员保护。管理员级相应于分段模型中的 0、1、2 级，它是赋给操作系统资源的级别。读/写（R/W）位使用户级的表或帧只读或可读/写。表 2-4 总结了 U/S 和 R/W 的访问属性。

表 2-4　段页机制下用户和管理员级访问权限

U/S	R/W	用　户	管 理 员
0	0	无	读/写
0	1	无	读/写
1	0	只读	读/写
1	1	读/写	读/写

页目录项指定的保护属性对页表中所有页帧都适用，但是页表指定的属性只适用于它定义的页帧。由于页帧存在两种保护属性，所以 32 位 CPU 能够保证较高级别（更加严格）的保护权限。图 2-27 中，目录和页表项中的其他一些位提供表或帧的使用统计信息。例如，P 位表示该项是否可用于页地址转换，P=1 表示该项有效，可用于地址转换，P=0 表示该项可能未定义，或者不在物理存储器中。如果试图访问 P=0 的页表或页帧，则发生缺页错误。该缺页错误由操作系统存储器管理程序来处理。

80386 也记录页表或页帧是否已被访问的事实。如果要对表或帧中的某个地址进行读/写访问，则先将表项的已访问位（A 位）设为 1，这表示该项已被访问过。对于页帧来讲，它还记录该访问是读操作还是写操作。D 位只用于页表项，如果相应的页帧中任一个地址执行了写操作则设置该位。在虚拟申请分页存储器系统中，操作系统可以检查这些位以确定当新页交换进物理存储器空间中时，存储器中的页是否需要对硬盘进行更新。最后三位为 AVL，可供编程者使用。

2.2.3　80x87 数学协处理器

80x87 是配合 80x86 微处理器用于完成浮点运算的数学协处理器。1980 年，8087 发布，然后为 80287、80387DX/SX 和 487SX 所替代。Intel 80486DX、Pentium 及其之后的处理器都在 CPU 中集成了数学协处理器。图 2-28 为数学协处理器的内部结构，主要分为两部分：控制单元和数字执行单元。

（1）控制单元

控制单元负责协处理器与微处理器的总线连接，两者同时监视指令流，若是 ESC 扩展指令，则由协处理器执行，否则由微处理器执行。

電子工業出版社

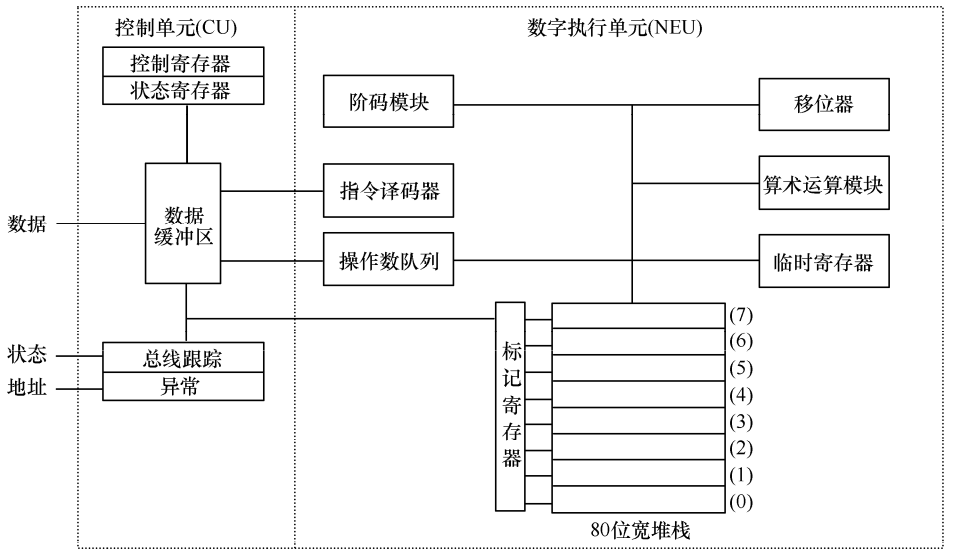


图 2-28　数学协处理器的内部结构

（2）数字执行单元

数字执行单元负责执行所有协处理器的任务，其中 8 个数据寄存器组成头尾相接后进先出的堆栈，用于存储算术指令的操作数和结果，对其数据的访问可采用特定寻址方式，但更多是使用堆栈的入栈和出栈操作。此外，数字执行单元还包括标记寄存器、状态寄存器和控制寄存器。

① 数据寄存器，有 8 个，将 80 位的扩展精度浮点数（编号 0～7）做成堆栈形式，如图 2-29 所示。数据驻留在内存时，可以是任何形式（带符号数、BCD 数、单精度或双精度数），但数据从内存移到浮点数据寄存器堆栈时，这些数据的格式将被转换为扩展精度浮点数。

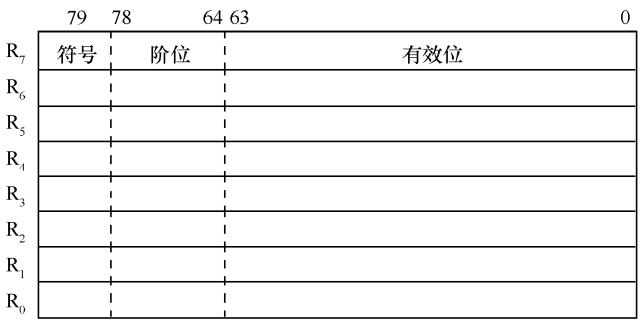


图 2-29　数学协处理器的数据寄存器组成

② 标记寄存器，表明浮点数据寄存器堆栈中每个单元的内容。图 2-30 给出了标记寄存器以及每个标记指示的状态。标记表明寄存器内容是否合法、是否为零、是否非法或为无穷大以及是否为空。

③ 状态寄存器，如图 2-31 所示。其中各位的含义如下。

B（Busy）：忙标志位，表明当前浮点运算单元正在执行一项任务。

C_3～C_0（Condition Code）：条件码位，表明浮点运算单元的工作条件（由这些码位的组合对应，但对于不同指令有不同的含义）。

電子工業出版社

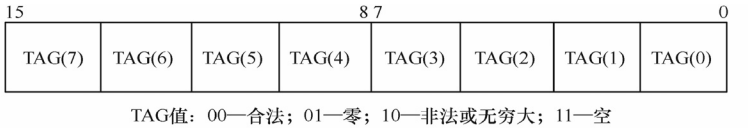


图 2-30　浮点运算的标记寄存器组成

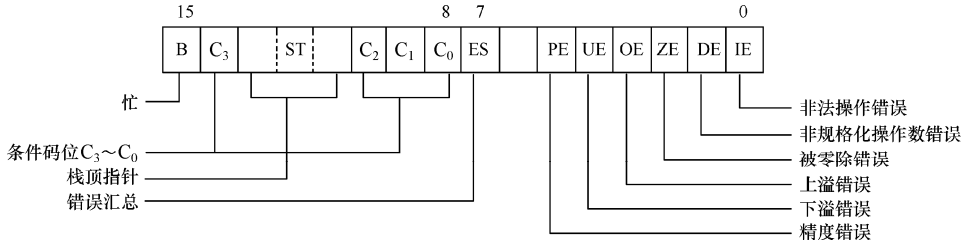


图 2-31　浮点运算的状态寄存器组成

ST（Top of Stack）：栈顶，表示当前寻址为栈顶的寄存器，通常是寄存器 R_0。

ES（Error Summary）：错误汇总位，当任何一个非屏蔽错误位被置位时，ES 被置位。

PE（Precision Error）：精度错误位，表明结果或操作数超过了设定的精度范围。

UE（Underflow Error）：下溢错误位，表明一个非 0 的结果太小，以致不能用当前精度来表示。

OE（Overflow Error）：上溢错误位，表明结果太大而不能被表示出来。

ZE（Zero Error）：被零除错误，表明当被除数是非无穷大和非零时，除数是零。

DE（Denormalized Error）：非规格化操作数错误，表明至少有一个操作数是非规格化的。

IE（Invalid Error）：非法操作错误，表明运算结果是不确定的形式，如 0÷0，+∞等，或者使用了 NAN（非数值量）作为操作数。

④ 控制寄存器，如图 2-32 所示，包括精度控制、舍入控制和无穷大控制，也可以屏蔽或开放与状态寄存器最右边 6 位对应的异常位。

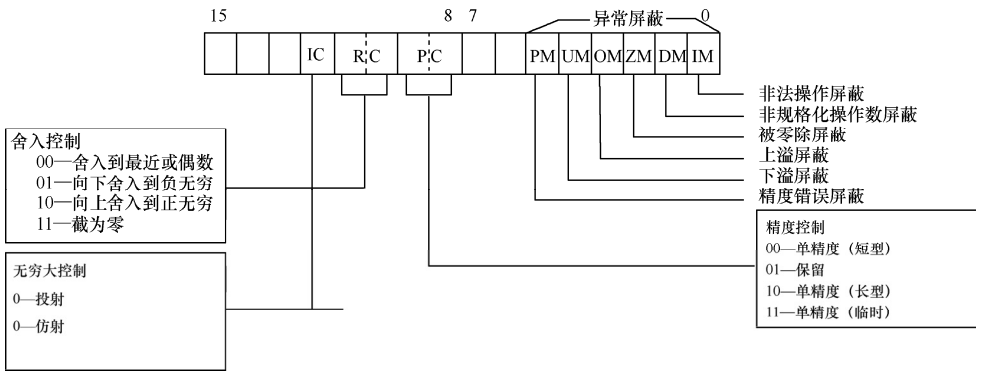


图 2-32　浮点运算的控制寄存器组成

以下是控制寄存器中各位及各组合位的功能。

IC（Infinity Control）：无穷大控制，选择是仿射无穷大还是投射无穷大。仿射允许正/负无穷大，而投射则假定无穷大为无符号数。

RC（Rounding Control）：舍入控制，确定舍入的类型。

PC（Precision Control）：精度控制，设置结果的精度。

异常屏蔽：决定异常错误是否影响状态寄存器的错误位。如果其中一个异常控制位被

電子工業出版社

置为 1，则相应的状态寄存器位被屏蔽。

2.2.4 Pentium CPU 内部结构

1. Pentium CPU 内部结构

Intel 公司于 1989 年 4 月推出新型 32 位处理器 80486，与 80386 相比，它的软件、硬件体系结构更先进，将浮点数学协处理器与指令、数据高速缓存集成在芯片上，这两个重要的改进使微机运行速度大大提高。80486 虽然在寻址方式、存储管理、数据类型等方面与 80386 相比的改变很少，但在相同主频下其处理速度比 80386 快 2～4 倍。

继 80486 后，1993 年 Intel 公司又推出新一代名为 Pentium 的微处理器（P5）。图 2-33 为 Pentium CPU 内部结构。Pentium 的中文名字为奔腾，奔腾是由希腊字 Pente（意思为 5）演变而来的，该处理器按原来代号顺序排列应取名为 80586，但为了取得商标注册，防止其他公司的兼容产品再以相同的名称命名，Intel 公司按照美国有关法律将这种新一代微处理器取名为 Pentium，并注册专有。

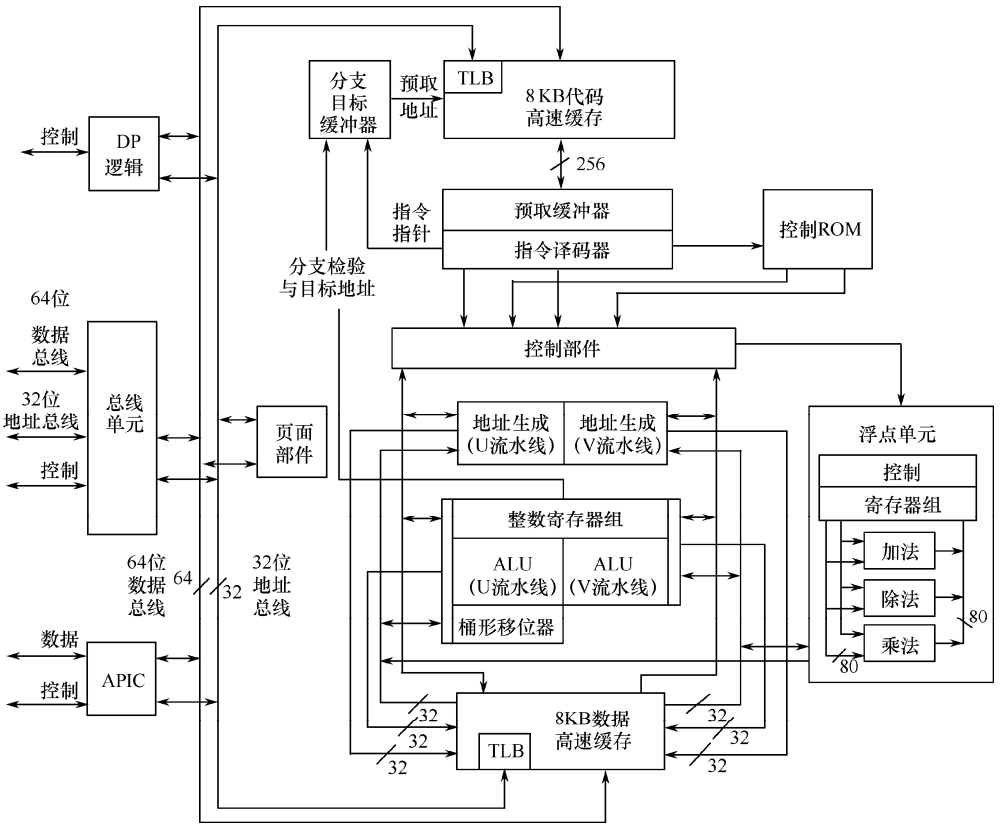


图 2-33　Pentium CPU 的内部结构示意

Pentium CPU 采用全新设计，具有 64 位数据线和 32 位地址线，与 80486 相比内部结构也有很大改进，但是依然保持了和 8086/8088、80286、80386、80486 的兼容性，在相同的工作模式上可以执行所有的 80x86 程序。片内存储管理单元（MMU）也与 386 和 486 兼容，可以在实地址方式引导下转入保护模式和虚拟 86 模式，其指令集包括了 80486 的所有指令，

并增加了新的指令。

Pentium CPU 主要由执行单元、指令 Cache、数据 Cache、指令预取单元、指令译码单元、地址转换与管理单元、总线单元以及控制器等组成，其中核心是执行单元（又称为运算器），它的任务是高速完成各种算术和逻辑运算，其内部包括两个整数算术逻辑运算单元（ALU）和一个浮点运算器，分别用来执行整数和实数的各种运算。

除执行单元外，其他单元都可认为是控制单元。其中主要包括指令预取单元、指令译码单元、地址转换与管理单元以及总线单元。总线单元为 Pentium CPU 提供了与周围其他硬件环境的接口，如从指令预取单元接收预取指令的请求，从执行单元接收传送数据的请求等，当有多个单元同时发出请求时，总线单元将进行协调，提供及时服务。总线单元的基本任务是使 Pentium CPU 能有效使用计算机中的总线。

2．Pentium CPU 的新特点

除了具有与 80x86 系列 CPU 完全兼容的特点，在体系结构上，Pentium CPU 还有如下一些新特点。

① Pentium CPU 片内高速缓存采用了分离式结构。Cache 分为两个成组相联的 8 KB 指令 Cache（用来存放程序代码）和 8 KB 的数据 Cache（用来存放数据），这种将片内高速缓存分开的做法使各自 Cache 能够加快速度，减少等待及搬移数据的次数和时间，从而提高了整体性能。

② Pentium CPU 采用了 RISC 技术。Pentium CPU 虽然属于 CISC 处理器，但在执行单元的设计中采用了较多的 RISC 技术，如具有超标量指令流水线功能。

③ Pentium CPU 具有高性能的浮点运算部件。其浮点运算部件完全重新设计，根据测试，其 FPU 速度是 80486 的 2 倍，其浮点操作已高度流水线化，并与整数流水线相结合。

④ Pentium CPU 具有分支指令预测功能。

⑤ 数据总线位宽增加。Pentium CPU 内部数据总线与 80486 一样都是 32 位（内部供程序控制用的寄存器组有 16 个，其中 32 位通用寄存器 10 个，32 位状态和控制标志寄存器 2 个，16 位段寄存器 6 个，另外还有更多的寄存器不再用来控制程序，而提供给 CPU 在操作过程中使用，这类寄存器有几十个），但 CPU 与内存进行数据交换的外部数据总线为 64 位，在一个总线周期内将数据传输量增加了一倍。另外，Pentium CPU 还支持多种类型的总线周期，在突发方式下，可以在一个总线周期内读入 256 B 的数据，其 64 位数据总线与内存数据交换的速率高达 528 MB/s，为 80486DX-50 的 3 倍还多。

⑥ 常用指令固化。在 Pentium CPU 中，把一些常用指令（如 MOV、INC、DEC、PUSH 等）改用硬件实现，不再使用微码操作，使指令的运行速度进一步加快。

⑦ 采用系统管理模式（SMM）。其最显著的应用就是电源管理，可以使 CPU 和系统外围部件在暂不工作时，休眠一定时间，然后在按下一键时唤醒它们，使之继续工作，通过使用管脚信号，SMM 甚至能完全控制整个系统，包括输入/输出和 RAM。

3．Pentium CPU 的内部寄存器

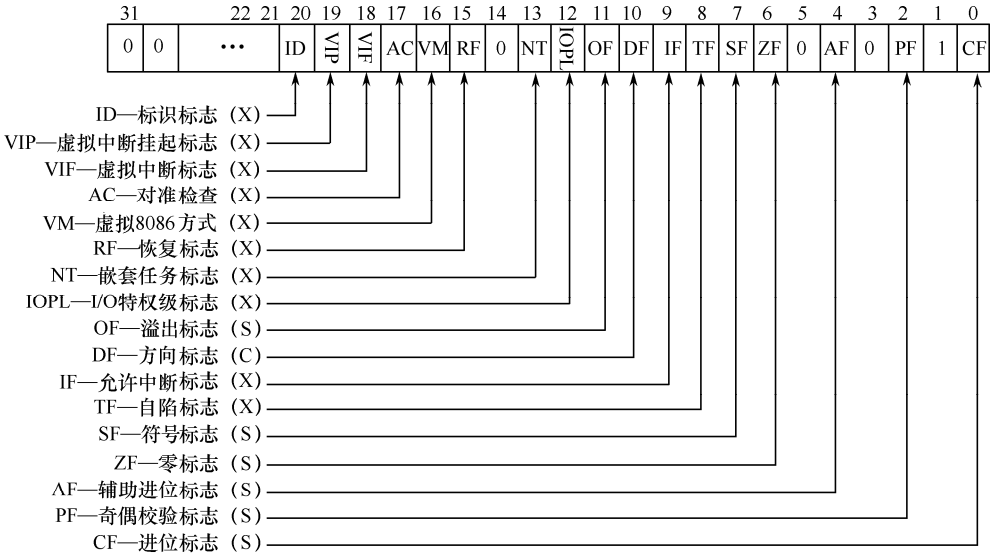
Pentium CPU 的内部寄存器按功能分为 4 类：基本寄存器、系统级寄存器、调试和模型特定寄存器、浮点寄存器。

（1）基本寄存器

基本寄存器包括通用寄存器、指令指针寄存器、标志寄存器和段寄存器，这些寄存器

電子工業出版社

均与80386寄存器基本相同，见图2-10。唯一有区别的是标志寄存器，如图2-34所示，Pentium CPU的标志寄存器比80386增加了4个标志位。



说明：S表示状态标志；C表示控制标志；X表示系统标志；0或1的位是Intel保留位

图2-34　Pentium CPU标志寄存器

① AC（Alignment Check）：对准检查标志。AC=1，且CR_0中的AM=1时，进行字、双字或四字的对准检查，如发现要访问的操作数未按边界对齐，会产生异常中断。

② VIF（Virtual Interrupt Flag）：虚拟中断标志。保护模式下中断允许标志位的副本。

③ VIP（Virtual Interrupt Pending）：虚拟中断挂起标志。用于在保护模式下提供中断信息，在多任务环境下，为操作系统提供虚拟中断标志和中断挂起信息。

④ ID（IDentification）：标识标志。用于指示CPU对CPUID指令的支持状态。CPUID指令为系统提供Pentium CPU的信息，如型号及制造商。

（2）系统级寄存器

系统级寄存器包括4个系统地址寄存器（与80386完全相同，见图2-15）和5个控制寄存器CR_0～CR_4。图2-35给出了Pentium CPU的控制寄存器。

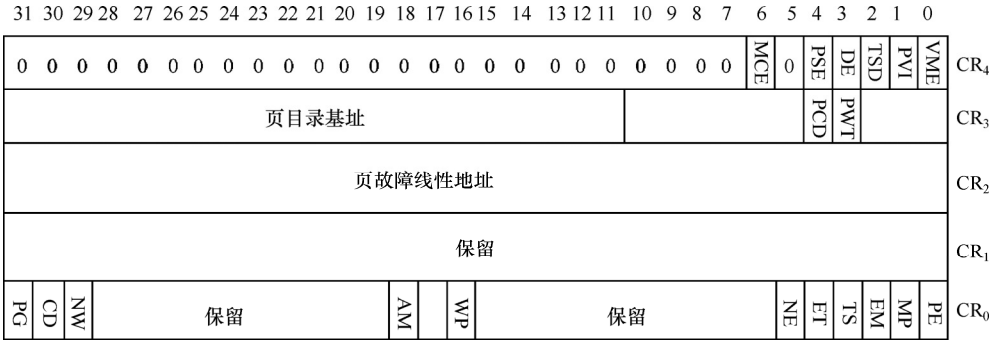


图2-35　Pentium CPU的控制寄存器

CR_0定义了11个控制位，比80386多定义了NE、WP、AM、NW、CD这5位，其含义如下。

① NE（Numeric Error）：数值异常控制位。控制是由中断向量 16 还是由外部中断来处理浮点数值异常。NE=1，任何未屏蔽的浮点异常将产生软件中断 16；NE=0，由外部中断来处理该异常。

② WP（Write Protect）：写保护位。WP=1，禁止特权级程序对只读页面的写操作；WP=0，允许只读页面由特权级写入。

③ AM（Alignment Mark）：对准屏蔽位。控制标志寄存器中对准检查位（AC）是否允许对准检查。AM=1，允许 AC 位对准检查；AM=0，则禁止 AC 位对准检查。

④ NW（Not Write through）：不通写控制位。选择片内 Cache 的操作模式。NW=1，禁止通写，写命中时不修改内存；NW=0，允许通写。

⑤ CD（Cache Disable）：Cache 禁止位。控制是否允许向未命中的 Cache 填充新数据。CD=1，当 Cache 未命中时，禁止填充 Cache；CD=0，当 Cache 未命中时，也允许填充 Cache。

CR_1：未定义的控制寄存器，供后期的微处理器使用。

CR_2：页故障线性地址寄存器，保存最后出现页故障的全 32 位线性地址。

CR_3：页目录基地址寄存器，保存页目录表的物理基地址。其中另外定义了 2 位，用于控制高速缓存，其含义如下。

① PWT（Page Write Through）：页写直达位。控制外部 Cache 写操作下的页直达方式。

② PCD（Page Cache Disable）：页高速缓存禁止位。PCD=1，表示禁止片内 Cache 存储当前页。

CR_4 中定义了 6 位，各位的含义如下。

① VME（Virtual Mode Extension）：虚拟方式扩展位。VME=1，表示允许虚拟 8086 方式的扩展。

② PVI（Protected mode Virtual Interrupt）：保护模式虚拟中断位。PVI=1，表示在保护模式下允许对虚拟中断标志的支持。

③ TSD（Time Stamp Disable）：时间标记计数器禁止位。TSD=1，且当前特权级不为 0 时，禁止读时间标记计数器指令 RDTSC；否则 RDTSC 将在所有特权级上执行。

④ DE（Debugging Extension）：调试扩展位。DE=1，允许 I/O 断点调试扩展。

⑤ PSE（Page Size Extension）：页尺寸扩展位。PSE=1，使能页面大小扩充为 4 MB。

⑥ MCE（Machine Check Enable）：机器检查允许位。MCE=1，允许机器检查异常。

（3）调试和模型特定寄存器

图 2-36　调试和模型特定寄存器

如图 2-36 所示，Pentium CPU 中设置了 8 个调试寄存器 DR_0～DR_7，它们为程序调试提供了硬件支持。其中 DR_0～DR_3 为线性断点寄存器，可保存 4 个断点地址，从而方便程序设计人员按照调试意图组合指令的执行。DR_4、DR_5 定义为 DR_6、DR_7 的别名。DR_6 用于保存断点状态，DR_7 用于控制断点设置。

专用模式寄存器 MSR（Model Special Register）是 Pentium CPU 特有的，用于跟踪和检查性能、测试和检查机器错误，它替换了 80386、80486 中的测试寄存器。Pentium CPU 使用 RDMSR 和 WRMSR 指令读/写专用模式寄存器，这两条指令都使用 ECX 寄存器给 CPU 传递寄存器号，使用 EDX:EAX 寄存器为 64 位宽读/写服务。Pentium CPU 专用模式

电子工业出版社

寄存器及其内容参如表 2-5 所示。

表 2-5　Pentium CPU 专用模式寄存器及其内容

地址（ECX）	寄存器名	大　小	说　明
00H	机器检查地址	64 位	引起异常的存储器地址
01H	机器检查类型	5 位	引起异常的存储周期类型
02H	测试寄存器 1	14 位	奇偶校验反向寄存器
03H	保留	—	—
04H	测试寄存器 2	4 位	指令 Cache 结束位
05H	测试寄存器 3	32 位	Cache 测试数据
06H	测试寄存器 4	32 位	Cache 测试标志
07H	测试寄存器 5	15 位	Cache 测试控制
08H	测试寄存器 6	32 位	TLB 测试线性地址
09H	测试寄存器 7	32 位	TLB 测试物理地址 A_{31}～A_{12}
0AH	测试寄存器 8	32 位	TLB 测试物理地址 A_{35}～A_{32}
0BH	测试寄存器 9	32 位	BTB 测试标志
0CH	测试寄存器 10	32 位	BTB 测试目标
0DH	测试寄存器 11	12 位	BTB 测试控制
0EH	测试寄存器 12	10 位	执行跟踪和转移预测
0FH	保留	—	—
10H	时间标记计数器	64 位	性能监测
11H	事件选择和控制	26 位	性能监测
12H	事件计数器 0	40 位	性能监测
13H	事件计数器 1	40 位	性能监测
14H	保留	—	—

（4）浮点寄存器

Pentium CPU 内嵌了算术运算协处理器，其浮点寄存器同 80x87，参见 2.2.3 节。

注意，Pentium CPU 的指令系统增加了 57 条新的多媒体扩展 MMX（MultiMedia Extensions）指令，专为动态视频、图像处理、音频合成等多媒体应用而设计。MMX 寄存器是利用别名（Aliasing）技术共享使用的浮点数据寄存器，即 MMX 寄存器（MM_0～MM_7）与浮点数据寄存器是相同的，如图 2-37 所示。

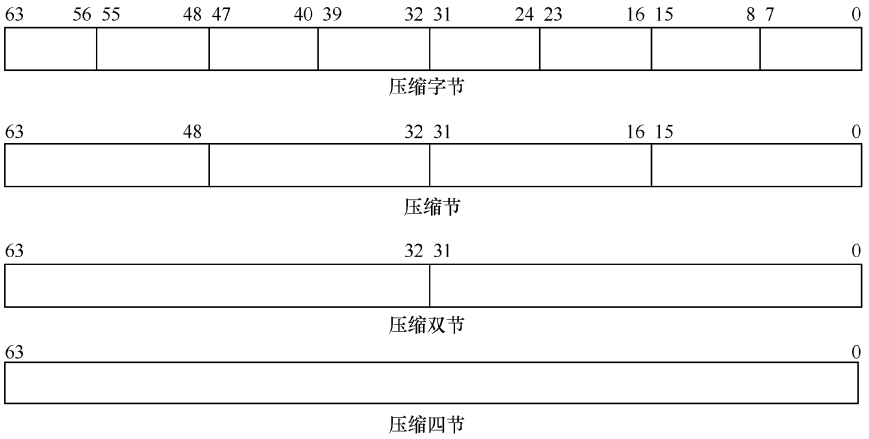


图 2-37　存储于 MMX 寄存器中的数据结构

注意，MMX 寄存器组为 64 位宽，并使用浮点数据寄存器组的最右边 64 位。MMX 体系结构引入新的紧缩数据类型是：8 个紧缩的、连续的 8 位的字节，4 个紧缩的、连续的 16

電子工業出版社

位的字，以及 2 个紧缩的、连续的 32 位的双字。MMX 寄存器与 64 位宽内存有相同的格式，它有两种数据存取方式，即 64 位存取方式和 32 位存取方式。

2.2.5 Pentium 系列其他微处理器

1．Pentium Pro

Pentium Pro 仍是 32 位微处理器，在 Pentium 的基础上增加的主要功能如下：

- 重新设计了微结构，是第一个采用 P6 结构的处理器。
- 集成了两级高速缓存。
- 采用了三路超标量结构和 14 级流水线深度，每个时钟周期可执行三条指令。
- 综合运用了 RISC 技术和 CISC 技术，采用了乱序执行和推测执行。

2．PentiumⅡ

Pentium Ⅱ在 Pentium Pro 的基础上扩展了 MMX 技术，其增加的主要功能如下：

- 采用多媒体增强技术，增加了 57 条 MMX 指令。
- 采用动态预测技术（也称随机推测执行技术）。
- 增加了 70 条 SSE 数据流单指令多数据扩展技术。
- 双独立总线技术。

PentiumⅡ的入门级处理器是减少了 L2（甚至取消 L2）的 Celeron。而 PentiumⅡ的高阶处理器是针对服务器设计的 Pentium Ⅱ Xeon。

3．Pentium Ⅲ

PentiumⅢ在 PentiumⅡ的基础上增加了 128 位的浮点寄存器，能同时处理 4 个单精度浮点数。增加了 70 条附加浮点多媒体指令，克服了不能同时处理 MMX 数据和浮点数据的缺陷，有效地增强了静止图像、视频处理的能力。

与 Pentium Ⅱ相同，PentiumⅢ也有低阶的 Celeron 版本和高阶的 Xeon 版本。

4．Pentium 4

Pentium 4 是第一个基于 Intel NetBurst 结构的处理器芯片，其主要特点如下：

- 采用超流水技术，流水线从 14 级提高到 20 级，有效地减少了执行时间。
- 采用快速执行引擎，为执行单元提供执行指令，防止执行单元停顿。
- 使用了包含 144 条指令的 SSE2 指令集，与 SSE 相比，使可操作的数据量加倍，从而提高了 CPU 执行指令的效率。

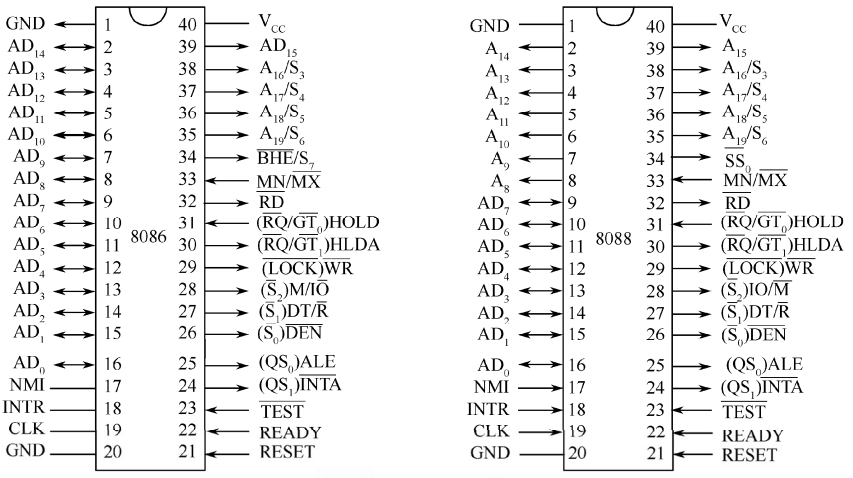
Pentium 4 同样有低阶的 Celeron 版本和高阶的 Xeon 版本。当其主频升为 4 GHz 时，遇到了无法解决的高功耗问题，这促使了功耗优化微处理器 Pentium M 的产生。

2.3 微处理器的主要引脚及功能

2.3.1 8086/8088 CPU 引脚功能

8086/8088 均为 40 条引脚（PIN）、双列直插式（DIP）封装，引脚分配如图 2-38 所示。

電子工業出版社



() 中为多CUP (最大) 模式中用

(a) 8086引脚　　(b) 8088引脚

图 2-38　8086/8088 引脚分配

8086/8088 的功能比 8 位 CPU 功能增强很多，因为：数据宽度从 8 位扩展到 16 位（8086）；直接寻址从 64 KB 扩展到 1 MB，采用段寄存器和存储器分段技术，利用某些引脚的双功能，可构成单 CPU 和多 CPU 系统。

为减少外部引脚的数目，在 8086/8088 CPU 内部设置了若干多路开关，使某些引脚具有多种功能，这些多功能引脚的功能转换分两种情况：① 分时复用，在总线周期的不同时钟周期内其功能不同；② 按工作模式定义引脚的功能，同一引脚在单 CPU（最小模式）和多 CPU（最大模式）下，连接不同的信号。

下面先说明 8088 引脚的功能，再从 8086 和 8088 的差别角度说明 8086 引脚的功能。

AD_7～AD_0：地址/数据总线，双向（入/出），三态。这是 8 根分时复用多功能引脚，担负着双重任务。在每个总线周期 T_1 状态用作地址总线低 8 位 A_7～A_0，输出访问存储器或 I/O 口地址，然后内部的多路转换开关将它们转换为数据总线 D_7～D_0，用来传送数据，直到总线周期结束。在 DMA 方式时，这些引脚成浮空状态。

A_{15}～A_8：地址总线，输出，三态。这些地址在整个总线周期内保持有效（即输出稳定的 8 位地址）。在 DMA 方式时，这些引脚为浮空状态。

A_{19}/S_6～A_{16}/S_3：地址/状态线，输出，三态。这是 4 根分时复用的多功能引脚，担负着双重任务，在每个总线周期 T_1 状态用作地址总线高 4 位。但对 I/O 口访问时，这些线为低电平，因为 I/O 口只用 16 位地址。在总线周期 T_2～T_4 期间，输出状态信息：S_6 总是低电平；S_5 是可屏蔽中断允许标志，可在每个时钟周期开始修改；S_4 和 S_3 表示当前访问存储器所用段寄存器，用来提供段地址。S_4 和 S_3 编码与段寄存器对应关系如表 2-6 所示。在 DMA 方式时，这些引脚为浮空状态。

表 2-6　S_4、S_3 编码与段寄存器的对应关系

S_4	S_3	性　能	对应段寄存器
0	0	数据交替	使用附加段寄存器 ES
0	1	堆栈操作	使用堆栈段寄存器 SS
1	0	代码	使用代码段寄存器 CS
1	1	数据	使用数据段寄存器 DS

NMI：非屏蔽中断请求，输入，上升沿有效。NMI 是上升沿触发的非屏蔽中断请求输入信号，不能通过软件进行屏蔽。只要该引脚上出现一个从低到高的电脉冲，就能使 CPU 现行指令结束，立刻进入中断响应，转去执行 NMI

電子工業出版社

中断处理。

INTR：可屏蔽中断请求，输入，高电平有效。INTR 是电平触发，CPU 在每条指令的最后一个时钟周期对 INTR 信号线采样，若发现 INTR 引脚信号为高电平，同时 CPU 内部中断允许标志 IF=1 时，CPU 就进入中断响应周期，CPU 响应中断时，暂停正在执行的主程序，从数据总线读取中断类型码，根据中断类型码从存储器的中断向量表里查找相应的中断服务程序的入口地址，转到该地址去执行中断服务程序，在中断结束后，再返回主程序的断点处继续执行程序。

若中断允许标志位 IF=0，即使 INTR 引脚信号为高电平，CPU 对外界送来的此中断请求信号也不予理睬。这样可以通过软件的方法使 IF=0，以实现屏蔽中断请求 INTR。

CLK：时钟信号，输入。CLK 为 CPU 和总线控制器提供定时基准。

表 2-7　初始化操作

复位操作影响部分	内　容
标志寄存器 FR	0000H
指令指针 IP	0000H
代码段寄存器 CS	FFFFH
数据段寄存器 DS	0000H
堆栈段寄存器 SS	0000H
附加段寄存器 ES	0000H
指令队列	空

RESET：复位信号，输入，高电平有效。当 RESET 为高电平时，CPU 将停止正在进行的操作，把内部标志寄存器 FR、段寄存器 DS、SS、ES 以及指令指针 IP 置 0，使指令队列清空，如表 2-7 所示。为保证对 CPU 进行可靠的初始化，RESET 高电平信号至少保持 4 个时钟周期，即当主频为 4.77MHz 时，复位时间必须大于 50μs。RESET 恢复低电平后，CPU 就从 FFFF0H 单元开始启动。

READY：准备就绪信号，输入，高电平有效。用来使 CPU 和慢速存储器或 I/O 设备之间速度实现匹配，完成数据传送。当被访问部件无法在 CPU 规定的时间内完成数据传输时，应使 READY 信号处于低电平，这时 CPU 进入等待状态，插入一个或几个等待周期 T_W 来延长总线周期。当被访问的部件可以完成数据传送时，READY 输入高电平，CPU 继续运行。慢速存储器或外部设备送出的 READY 信号必须先经过 8284 时钟发生器，与时钟脉冲同步后，再送入 CPU 的 READY 引脚，可满足定时要求，避免动作失误。

$\overline{\text{TEST}}$：测试信号，输入，低电平有效。当执行 WAIT 指令时，每隔 5 个时钟周期，CPU 就对 $\overline{\text{TEST}}$ 信号进行采样，若 $\overline{\text{TEST}}$ 为高电平，就使 CPU 重复执行 WAIT 指令并处于等待状态，一直等到它变为低电平时，CPU 才脱离等待状态，继续执行下一条指令。通常，$\overline{\text{TEST}}$ 引脚和 WAIT 指令结合使用，主要用于多处理器环境。例如，在具有协处理器 8087 的系统中，将其 BUSY 引脚接 $\overline{\text{TEST}}$。每当主 CPU 执行 WAIT 指令时，反复采样 $\overline{\text{TEST}}$ 信号，直到协处理器的 BUSY 信号使 $\overline{\text{TEST}}$ 变为低电平为止。

$\overline{\text{INTA}}$：中断响应信号，输出，三态，低电平有效，是 CPU 响应外部中断请求后，发给请求中断的设备回答信号。在每个中断响应周期 T_2、T_3、T_W 期间，它变为有效低电平，通知中断源送出中断矢量码。所以外部设备可以用此信号作为读取矢量码的选通信号。

ALE：地址锁存允许信号，输出，高电平有效。当地址/数据总线分时传输地址信息时，ALE 用作把地址信号锁存入 8282/8283 或 74LS373 锁存器的锁存控制信号。在总线周期 T_1 期间，ALE 出现高电平，而 ALE 的下跳沿将出现在地址/数据总线上的地址信息锁存入锁存器。在 DMA 方式中，ALE 引脚不能浮空。

$\overline{\text{DEN}}$：数据允许，输出，三态，低电平有效。在单 CPU 系统中，如果用 8286/8287 作为数据总线的双向驱动器，则用 $\overline{\text{DEN}}$ 作为驱动器的选通信号。在每个存储器或 I/O 访问周期以及中断响应周期，$\overline{\text{DEN}}$ 变为有效低电平。在 DMA 方式时，它处于浮空状态。

電子工業出版社

DT/$\overline{R}$：数据发送/接收控制，输出，三态。在单 CPU 系统中，如果用 8286/8287 作为数据总线的双向驱动，这时要用 DT/$\overline{R}$ 来控制 8286/8287 的数据传送方向。DT/$\overline{R}$=1 时，CPU 发送数据；DT/$\overline{R}$=0 时，CPU 接收数据。在 DMA 方式时，它处于浮空状态。

IO/$\overline{M}$：外设/内存访问控制，输出，三态。输出高电平时，表示总线周期为 I/O 访问周期；输出低电平时，表示总线周期为存储器访问周期。在 DMA 方式时，它为浮空状态。

$\overline{WR}$：写信号，输出，三态，低电平有效。$\overline{WR}$ 信号有效时，表示 CPU 正在执行写存储器或写 I/O 口操作。由 IO/$\overline{M}$ 信号的状态决定数据是写入存储器还是写入 I/O 端口。在 DMA 方式时，它处于浮空状态。

$\overline{RD}$：读信号，输出，三态，低电平有效。$\overline{RD}$ 信号有效时，表示 CPU 在读存储器或 I/O 口的数据。IO/$\overline{M}$ 为低电平时，CPU 读存储器。在 DMA 方式时，它处于浮空状态。

HOLD：保持请求信号，输入，高电平有效。当 DMA 操作或外部处理器要求通过总线传送数据时，HOLD 信号为高电平，表示外界请求现有主 CPU 让出对总线的控制权。

HLDA：保持响应信号，输出，高电平有效。当 CPU 同意让出总线控制权时，输出 HLDA 高电平信号，通知外界可以使用总线。同时现有主 CPU 所有“三态”线都进入浮空状态；当 HOLD 变为低电平时，现有主 CPU 把 HLDA 变为低电平，此时将重新获得总线控制权。

$\overline{SS_0}$：状态信号，输出，三态。它在逻辑上等同于多 CPU 系统中的 $\overline{S_0}$。$\overline{SS_0}$ 只用在单 CPU 系统中。$\overline{SS_0}$、IO/$\overline{M}$、DT/$\overline{R}$ 组合表示的单 CPU 系统总线周期状态对应关系如表 2-8 所示。在多 CPU 系统下，$\overline{SS_0}$ 总是输出高电平。

MN/$\overline{MX}$：单 CPU/多 CPU 方式控制，输入。该引脚的电平，决定了系统是构成单 CPU（最小模式）还是多 CPU（最大模式）系统。当 MN/$\overline{MX}$=V_{CC} 时，按单 CPU（最小模式）工作，这时 8088 的 24～31 引脚功能如上所述；若 MN/$\overline{MX}$=GND，系统按多 CPU（最大模式）工作，8088 的 24～31 引脚定义如图 2-38 括号内所示。

下面的引脚在单 CPU 和多 CPU 系统下具有不同的功能。IBM PC/XT 具有协处理器 8087，硬件扩展功能强，所以通常设计 8088 工作于多 CPU（最大模式）系统方式下，下面主要介绍图 2-38 括号内引脚的功能。

$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$：总线周期状态标志，输出、三态，低电平有效。这三条引脚状态信号的不同组合，表示 CPU 总线周期的操作类型。这些信号送入 8288 总线控制器对应输入端，8288 利用这些信号的不同组合，产生访问存储器或 I/O 的控制信号或中断响应信号。这些信号在每个总线周期 T_1 有效，而在 T_3 或 T_W 后半周期，且 READY 为高电平时，则返回无效状态，表示总线周期结束。$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$ 的编码和总线周期的关系如表 2-9 所示。在 DMA 方式时，它们进入高阻状态。

表 2-8　单 CPU 系统总线周期状态

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SS_0}$	操　作
1	0	0	中断响应
1	0	1	读 IO/口
1	1	0	写 I/O 口
1	1	1	暂停
0	0	0	取指令
0	0	1	读存储器
0	1	0	写存储器
0	1	1	无效

表 2-9　总线周期状态标志

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	操作类型
0	0	0	中断响应
0	0	1	读 I/O 口
0	1	0	写 I/O 口
0	1	1	暂停
1	0	0	取指令操作码
1	0	1	读存储器
1	1	0	写存储器
1	1	1	无效状态

電子工業出版社

$\overline{RQ}/\overline{GT_0}$、$\overline{RQ}/\overline{GT_1}$：请求/允许控制信号，双向，三态，低电平有效。这两个引脚供外部主控设备用来请求获得总线控制权。首先由外部主控设备向 8088 输入请求使用总线的信号（保持信号 HOLD），然后在同一条线上，8088 输出允许外部主控设备使用总线的回答信号（保持响应 HLDA）两条控制线可同时接两个外部主控设备，但 $\overline{RQ}/\overline{GT_0}$ 的优先权高于 $\overline{RQ}/\overline{GT_1}$。若两者同时向 8088 申请，先响应 $\overline{RQ}/\overline{GT_0}$ 上的申请。

请求/允许控制信号工作时序如图 2-39 所示。

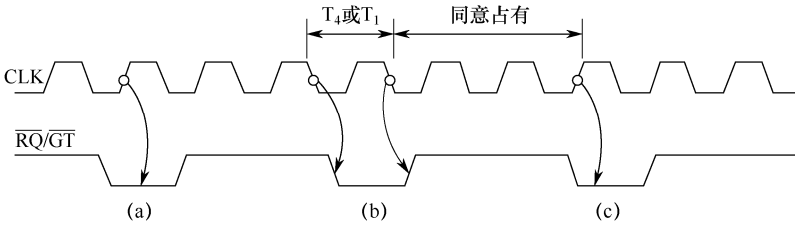


图 2-39　请求/允许控制信号工作时序

① 由其他主控制设备在 $\overline{RQ}/\overline{GT}$ 引脚向 8088 发出一个时钟周期宽的负脉冲，表示请求占用总线。

② 在当前总线周期 T_4 或下个总线周期 T_1 期间，8088 输出一个宽度为一个时钟周期的负脉冲，通知主控设备，表示 8088 同意让出总线，并从下一个时钟周期开始进入“同意占有”总线状态。在此期间 8088 所有三态线（除了 $\overline{RQ}/\overline{GT}$ 线）均置成浮空状态。

③ 主控设备申请总线控制权结束，送出一个时钟周期宽的负脉冲给 8088，表示请求占用总线周期已经结束，8088 可以在下一个时钟脉冲上升沿收回总线控制权。

$\overline{LOCK}$：封锁信号，输出，三态，低电平有效，用来封锁外部主控设备的请求。当 8088 的 $\overline{LOCK}$ 信号为低电平时，外部主控设备不能占用总线。该信号是用指令在程序中设置的。如果一条指令加上指令前缀 LOCK，则 8088 执行这条指令时，$\overline{LOCK}$ 引脚为低电平，并保持到指令结束，防止该指令执行过程中被打断。在 DMA 工作方式时，$\overline{LOCK}$ 为浮空状态。

QS_1、QS_0：指令队列状态，输出，高电平有效。QS_1、QS_0 的不同编码状态反映了 CPU 内部当前的指令队列状态，以便外部主控设备对 8088 进行跟踪。QS_1、QS_0 不同编码和指令队列的状态如表 2-10 所示。

8086 与 8088 引脚有以下区别：

① 8086 有 16 条地址/数据引脚 AD_{15}～AD_0。这是因为 8086 的数据总线是 16 位，所以 16 条引脚地址与数据分时复用；而 8088 只有 AD_7～AD_0 分时复用。

② 8086 的 34 引脚为 $\overline{BHE}/S_7$（8088 为 $\overline{SS_0}$），是高 8 位数据总线的允许和状态信息复用引脚。$\overline{BHE}$ 可以看成一条附加的地址总线，用来访问存储器的高字节，而 A_0 用来访问存储器的低字节。所以 $\overline{BHE}$ 通常作为接在高 8 位数据总线上的设备片选信号，而 A_0 作为接在低 8 位数据总线上的设备片选信号。$\overline{BHE}$ 和 A_0 的状态与数据总线传送的数据之间的关系如表 2-11 所示。

在总线周期 T_1 期间 $\overline{BHE}$ 输出有效信息，而在 T_2、T_3、T_W、T_4 期间，这条引脚输出 S_7 状态信息，S_7 低电平有效。在 DMA 工作方式，它为浮空状态。

③ 8086 的 28 引脚为 $M/\overline{IO}$，存储器/输入输出信号，输出，三态。当 $M/\overline{IO}$=1 时，表示访问存储器；当 $M/\overline{IO}$=0 时，表示访问 I/O 端口，与 8088 第 28 引脚（$IO/\overline{M}$）逻辑定义正好相反。

電子工業出版社

表 2-10　指令队列状态

QS_1	QS_0	意　义
0	0	无操作
0	1	取指令队列中第一操作码
1	0	队列空
1	1	取指令队列中后续字节

表 2-11　$\overline{BHE}$ 和 A_0 编码与数据总线数据传送的关系

$\overline{BHE}$	A_0	数据传送状态
0	0	传送 16 位 D_{15}～D_0
0	1	传送高 8 位 D_{15}～D_8
1	0	传送低 8 位 D_7～D_0
1	1	无操作

2.3.2　80386 CPU 主要引脚功能

80386 通常有两个型号：80386DX 和 80386SX。80386DX 采用 132 引脚的 PGA（引脚栅格阵列）封装，具有 32 位数据和地址总线；80386SX 是一个缩减总线型的 80386，具有 16 位数据总线和 24 位地址总线。下面以 80386DX 为例介绍 80386 的主要引脚功能。80386DX 的引脚分配图如图 2-40 所示。需要说明的是，为便于书写，从 80386 开始，其产品数据手册中把低电平或负脉冲有效的表示方法由习惯上的加上划线改成后跟“#”，如通常的 M / $\overline{IO}$ 信号变成了 M/IO#，$\overline{BE_3}$ 等同于 BE_3 #。

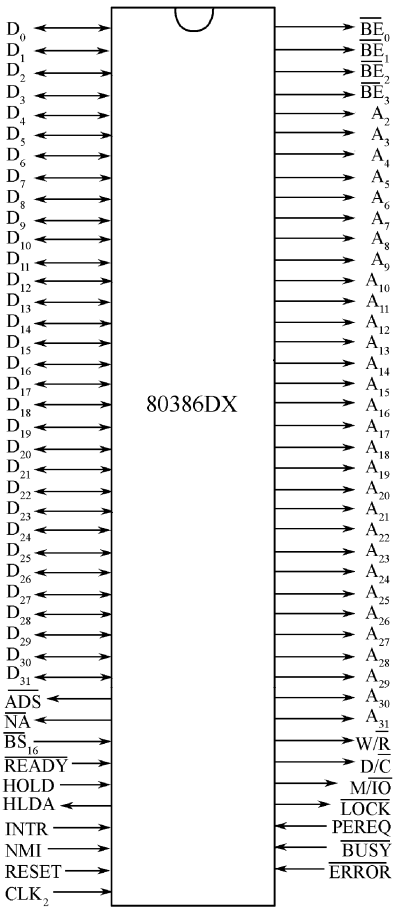


图 2-40　80386DX 引脚分配

A_{31}～A_2：地址总线，用来访问 80386 存储系统中的 1G×32 位存储单元。A_1、A_0 被编码为总线允许信号（$\overline{BE_3}$～$\overline{BE_0}$），以便选择 32 位存储单元中 4 字节的任何一个或几个。

D_{31}～D_0：数据总线，用于在 CPU 和存储器及 I/O 之间传送数据。

$\overline{BE_3}$～$\overline{BE_0}$：存储体使能信号，选择字节、字或双字数据。这些信号由 CPU 内部根据地址线 A_1 和 A_0 产生。

$\overline{ADS}$：地址数据选通引脚。当 80386 发出有效的存储器地址或 I/O 地址时，产生该信号。

RESET：复位引脚。用来初始化 80386，使其从内存 FFFFFFF0H 处开始执行软件，80386 被复位为实模式。

$\overline{BS_{16}}$：16 位总线宽输入引脚。用来选择 32 位或 16 位数据总线。

M / $\overline{IO}$：存储器及 I/O 选择引脚，为 1 时选择存储器设备，为 0 时选择 I/O 设备。在 I/O 操作期间，地址总线包含了 16 位的地址，即 A_{15}～A_2 及低 2 位。

W / $\overline{R}$：写/读引脚。W / $\overline{R}$ 为 1，表明当前总线周期是写周期；为 0，则表明是读周期。

CLK_2：两倍频时钟引脚，由 2 倍于 80386 工作频率的时钟信号驱动。

INTR：中断请求信号。外部电路使用该信号请求中断。

NMI：非屏蔽中断输入引脚。请求非屏蔽中断，与早期 CPU 的作用一样。

$\overline{READY}$：就绪引脚，该引脚控制为延长存储器或 I/O 访问时间而插入的等待状态的个数。

電子工業出版社

$\overline{\text{LOCK}}$：总线锁定引脚。当指令带有 LOCK 前缀时，该引脚变为逻辑 0，经常用于 DMA 访问期间。

D/$\overline{\text{C}}$：数据/控制输出引脚。D/$\overline{\text{C}}$为 1 时，表示数据总线含有来自或送到存储器或 I/O 的数据；为 0，则表明 CPU 正在执行中断响应或已停机。

$\overline{\text{NA}}$：下一地址信号。该信号使 80386 在当前总线周期输出下一条指令或数据的地址，它通常用在流水线操作中。

HOLD：总线保持引脚。用来请求 DMA 操作。

HLDA：总线保持响应引脚。用来表明 80386 当前处于总线保持状态。

PEREQ：协处理器请求信号。该信号要求 80386 放弃总线控制并直接连到 80387 算术协处理器上。

$\overline{\text{BUSY}}$：忙输入信号。由 WAIT 或 FWAIT 指令使用，等待协处理器变为空闲。

2.3.3 Pentium CPU 主要引脚功能

Pentium CPU 有两个版本：全功能型 Pentium 和称为 Pentium OverDrive 的 P24T 型。全功能型 Pentium 为 237 引脚 PGA 封装，数据总线 64 位，地址总线 32 位。P24T 型具有 32 位数据总线，可以兼容地插到有 P24T 插座的 80486 机器中。早期型号的 Pentium 需要+5 V 电源，电流较大，导致功耗很大；后期型号的 Pentium，主频为 90 MHz 以上，采用 3.3 V 电源供电，以减少电流消耗，同时配有良好通风效果的散热器来保证芯片不致过热。Pentium 微处理器引脚示意如图 2-41 所示。

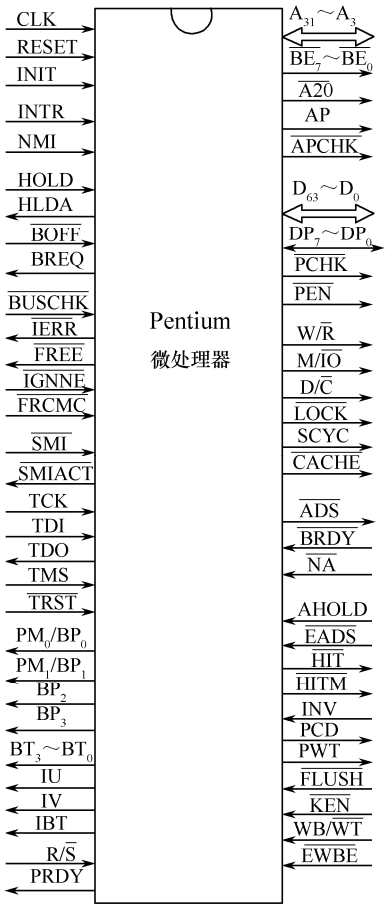


图 2-41　Pentium 微处理器引脚示意

Pentium 主要引脚的功能说明如下。

A_{31}～A_3：地址总线引脚，用于寻址 Pentium 存储系统中任何 512K×64 位存储空间，A_2、A_1、A_0 在总线允许（$\overline{\text{BE}_7}$～$\overline{\text{BE}_0}$）中被编码用于选择 64 位宽存储单元中的任意或全部 8 字节。

$\overline{\text{ADS}}$：地址数据选通信号。当 Pentium CPU 发出一个有效的存储器地址或 I/O 地址时，该信号变为有效。

$\overline{\text{A20}}$：地址 A_{20} 屏蔽输入引脚，用于在实模式中通知 Pentium CPU 进行地址扩展。

$\overline{\text{BE}_7}$～$\overline{\text{BE}_0}$：存储体允许信号，用于选择访问单字节、字、双字、四字数据。这些信号在 CPU 内由地址 A_2、A_1、A_0 产生。

D_{63}～D_0：数据总线信号。在 CPU、内存和 I/O 系统间进行字节、字、双字、四字数据的传输。

RESET：复位信号。初始化 Pentium CPU，使其从内存 FFFFFFF0H 处开始执行，Pentium CPU 被复位为实模式，最高位的 12 条地址线保持逻辑 1（FFFH），直至执行一个远程调用或远跳转，这样可使它与早期的 CPU 兼容。硬件复位后 Pentium CPU 的状态如表 2-12 所示。

電子工業出版社

表 2-12　RESET 后的 Pentium CPU 状态

寄 存 器	RESET 后的值	寄 存 器	RESET 后的值
EAX	0	GDTR，TSS	0
EDX	0500xxxxH	CR_0	60000010H
EBX，ECX，ESP，EBP，ESI，EDI	0	CR_2，CR_3，CR_4	0
EFLAGS	2	DR_0～DR_3	0
EIP	0000FFF0H	DR_6	FFFF0FF0H
CS	F000H	DR_7	00000400H
DS，ES，FS，GS，SS	0	—	

注：xxxx=Pentium 版本号。

$\overline{\text{BRDY}}$：猝发就绪信号。该信号通知 Pentium CPU 外部系统已从数据总线连接中取得数据。可用于向 Pentium CPU 时序中插入等待状态。

BREQ：总线请求输出信号，指示 Pentium CPU 已产生一个总线请求。

CLK：时钟引脚。由具有当前 Pentium CPU 工作频率的时钟信号驱动。

D/$\overline{\text{C}}$：数据/控制信号。该信号为逻辑 1 时表明数据总线上含有来自存储器或 I/O（或写往存储器或 I/O）的数据，为逻辑 0 时则表明 CPU 处于停机状态，或者正在执行一个中断请求。

$\overline{\text{FLUSH}}$：清 Cache 输入信号。使 Cache 清除所有写回行并使内部 Cache 无效。如果在进行复位时该信号为 0，则 Pentium CPU 进入 Cache 测试模式。

$\overline{\text{CACHE}}$：Cache 输出信号。指示当前 Pentium CPU 周期可对数据进行缓存。

$\overline{\text{HIT}}$：命中信号。表明在查询方式中内部 Cache 包含了合法数据。

$\overline{\text{HITM}}$：命中修改信号。表明在查询周期中发现了一个修改过的 Cache 行，此输出信号用于在已修改过的 Cache 行写回到存储器之前禁止其他单元访问数据。

WB/$\overline{\text{WT}}$：写回/写直达信号，为 Pentium CPU 数据 Cache 选择相应操作。

$\overline{\text{KEN}}$：Cache 使能输入信号。允许使用内部高速缓存。

HOLD：保持信号。请求一个 DMA 操作。

HLDA：保持响应信号。指示 Pentium CPU 当前处于保持状态。

INTR：中断请求信号。外部电路用该信号进行中断请求。

INV：无效输入信号。决定一个查询后的 Cache 行状态。

IU：U 管道指令完成输出信号。表明 U 管道中的指令已完成。

IV：V 管道指令完成输出信号。表明 V 管道中的指令已完成。

$\overline{\text{LOCK}}$：锁定信号。指令带有 LOCK 前缀时该信号变为逻辑 0。该信号通常用在 DMA 访问中。

M/$\overline{\text{IO}}$：存储器及 I/O 选择引脚。M/$\overline{\text{IO}}$ 为 1 时选择存储器设备，为 0 时选择 I/O 设备。在 I/O 操作期间，地址总线包含了 16 位的地址，即 A_{15}～A_3 及低 3 位。

W/$\overline{\text{R}}$：写/读引脚。W/$\overline{\text{R}}$ 为 1 时表明当前总线周期是写周期，为 0 时表明是读周期。

$\overline{\text{NA}}$：下一地址信号。表明外部存储器系统已准备好接收新的总线周期。

NMI：非屏蔽中断信号。请求一个非屏蔽中断。

$\overline{\text{SMI}}$：系统管理中断输入信号。使 Pentium CPU 进入系统管理运行模式。

$\overline{\text{SMIACT}}$：系统管理中断激活输出信号。表明 Pentium CPU 正工作在系统管理模式。

TCK：可测试性时钟输入信号。根据 IEEE 1149.1 边界扫描接口选择时钟操作。

電子工業出版社

TDI：测试数据输入信号。测试由 TCK 信号扫入 Pentium CPU 的数据。

TDO：测试数据输出信号。获得由 TCK 移出的 Pentium CPU 测试数据和指令。

TMS：测试方式选择输入信号。在测试模式中控制 Pentium CPU 操作。

$\overline{\text{TRST}}$：测试复位输入信号。使测试模式被复位。

2.4 系统总线与典型时序

2.4.1 CPU 系统总线及其操作

如 1.1.1 节所述，微机系统中各部分之间都是通过总线联系在一起并进行信息传输的。总线是在模块与模块之间或者设备与设备之间传输信息的一组公用信号线，是系统在主控器的控制下，将发送器发出的信息准确地传送给某个接收器的信号通路。总线作为所有模块或设备共同使用的“公路”，每个模块或设备都通过门电路与总线中相应的信号线相连。作为发送器的模块或设备可以通过驱动器把要输出的信号送到总线中相应信号线上进行传输；作为接收器的模块或设备则在适当时刻打开接收总线信号的缓冲器或寄存器，把总线相应信号线上的信号接收进来。

CPU 系统中有 3 种总线：① 地址总线，为存储器和 I/O 设备提供存储器地址或 I/O 端口号；② 数据总线，在系统中用于 CPU 与存储器及 I/O 之间传输数据；③ 控制总线，为存储器和 I/O 提供控制信号。数据总线和地址总线比较简单，型号不同但位数相同的 CPU，其特性基本相同，功能也比较单纯，但控制总线因 CPU 型号不同而相差很大。控制总线的不同特性，决定了各种 CPU 的不同接口特点。

微机系统中的各种操作，包括 CPU 把数据写入输出端口、从输入端口读入数据，CPU 把数据写入存储器，从存储器读出数据，DMA 访问操作等，本质上都是通过总线进行信息交换，这些操作统称为总线操作。在同一时刻，总线上只允许一对模块进行信息交换。当有多个模块都要使用同一总线进行信息传输时，只能采用分时方式，轮流使用总线，即将总线时间分成很多段，每段时间可以完成模块之间一次完整的信息交换，通常称为一个数据传输周期或一个总线操作周期。完整的总线操作周期一般分成以下 4 个阶段。

① 总线请求和仲裁阶段。需要使用总线的主模块提出请求，由总线仲裁机构确定把下一个传输周期的总线使用权分配给哪一个请求源。

② 寻址阶段。取得使用权的主模块通过总线发出本次要访问的从模块的存储器地址或 I/O 端口地址，启动参与本次传输的从模块。

③ 数据传输阶段。主模块和从模块进行数据交换。在主模块发出的控制信号作用下，数据由源模块发出，经数据总线传送到目的模块。

④ 结束阶段。主、从模块的有关信息从系统总线上撤除，让出总线，以便其他模块能使用。

为了确保这 4 个阶段正确推进，必须施加总线操作控制。对于只有一个主模块的单 CPU 系统，总线始终归它所有，所以不存在总线的请求、分配和撤除等问题，其数据传输周期只需要寻址和传输数据两个阶段。图 2-42 为 8086/8088 的典型 BIU 总线周期，由 4 个时钟周期组成，称为 T_1、T_2、T_3、T_4 状态。其中 T_1 期间输出地址，T_2 期间总线转向，T_3 期间完成数据传输，T_4 期间总线周期结束。

電子工業出版社

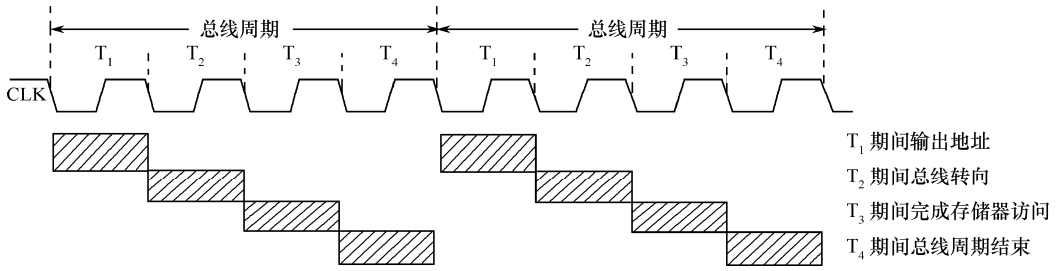


图 2-42　8086/8088 的典型 BIU 总线周期

2.4.2　基本总线操作时序

8086/8088 CPU 的操作都是在系统时钟 CLK 控制下严格定时的。按照一般概念，一条指令从取出到执行完毕所持续的时间称为指令周期。指令周期由若干机器周期组成。一个机器周期就是完成某一独立操作所持续的时间，如取指令操作码、存储器读/写等。一个机器周期由几个时钟周期组成。时钟周期（或 T 状态）是两个时钟脉冲上升沿之间的持续时间，它是 CPU 最小的定时单位。但是 8086/8088 CPU 由总线接口部件 BIU 和执行部件 EU 两部分组成。在 2.2.1 节中已述及，EU 和 BIU 是两个独立部件，而且是独立操作。EU 在执行指令过程中，必须读/写存储器或 I/O 端口中的操作数时，由 EU 向 BIU 提出申请，BIU 响应 EU 的请求去执行某个访问存储器或 I/O 端口的读/写机器周期，如上节提及的，这个机器周期又称为总线周期。

接下来以 8086/8088 为例，说明基本总线操作时序。

（1）8086 存储器读周期时序（如图 2-43 所示）

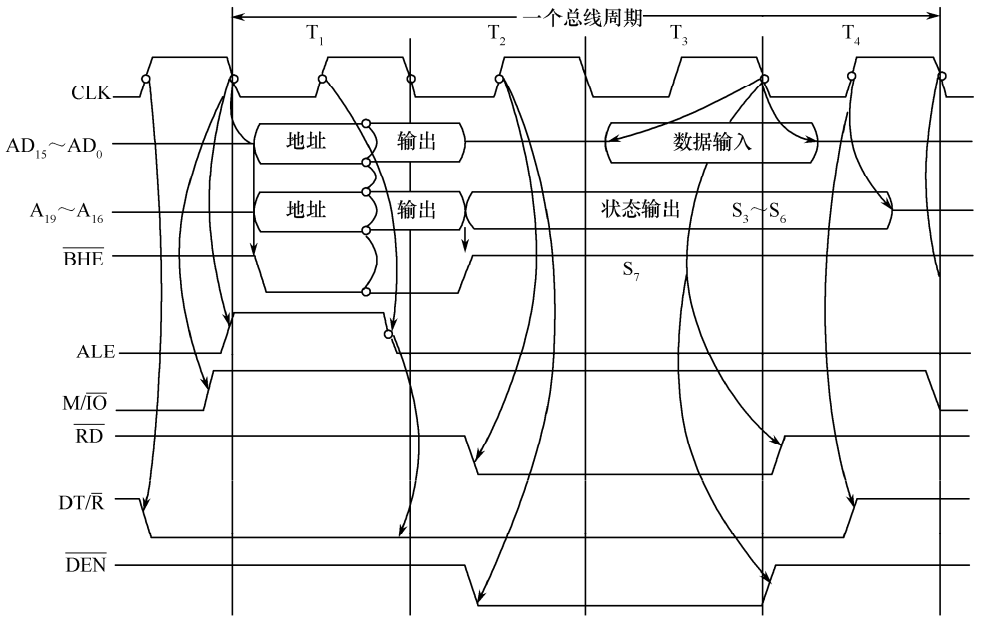


图 2-43　8086 存储器读周期时序

在 T_1 时钟周期开始，先用 $M/\overline{IO}$ 信号指出 CPU 是读内存还是 I/O 端口，所以 $M/\overline{IO}$ 信号在 T_1 时必须有效。若是从存储器读数据，则 $M/\overline{IO}$ 为高；若是从 I/O 端口读数据，则为

电子工业出版社

低。$M/\overline{IO}$ 信号的有效电平一直保持到整个总线周期结束，即 T_4 状态。

此外，在 T_1 开始 BIU 把访问存储器（或 I/O 端口）的物理地址 A_{19}/S_6～A_{16}/S_3 及 AD_{15}～AD_0，连同 $\overline{BHE}$ 送至总线上，即从 T_1 开始，20 位（16 位）地址信息通过这些引脚送到存储器（或 I/O 端口）。

地址信息必须被锁存起来，这样才能在总线周期的其他状态，复用这些引脚传送数据和状态信息。为实现对地址的锁存，CPU 在 T_1 状态从 ALE 引脚上输出一个正脉冲作为地址锁存信号。在 ALE 的下降沿到来之前，$M/\overline{IO}$ 信号、地址信号均已有效。8282 锁存器是用 ALE 的下降沿对地址进行锁存。

$\overline{BHE}$ 信号也在 T_1 状态通过 $\overline{BHE}/S_7$ 引脚送出，它用来表示高 8 位数据总线上的信息可以使用。

需要指出的是，系统中常常接有数据总线收发器 8286，要用到 $DT/\overline{R}$ 和 $\overline{DEN}$ 作为控制信号。为此，在 T_1 状态，$DT/\overline{R}$ 端输出低电平，表示本总线周期为读周期，即让数据总线收发器 8286 接收数据。

在 T_2 状态，地址信号消失，此时 AD_{15}～AD_0 进入高阻状态，以便为读入数据做好准备；而 $\overline{BHE}/S_7$ 和 A_{19}/S_6～A_{16}/S_3 引脚上输出状态信息 S_7～S_3。

$\overline{DEN}$ 信号在 T_2 状态变为低电平，在系统中接有 8286 总线收发器时，获得数据允许信号。

$\overline{RD}$ 信号在 T_2 状态变为低电平，被地址信号选中的内存单元（或 I/O 端口），才被 $\overline{RD}$ 信号从中读出数据，将其送到数据总线上。

在 T_3 状态，将内存单元（或 I/O 端口）的数据送到数据总线上，CPU 通过 AD_{15}～AD_0 接收数据。

在 T_4 状态和前一个状态交界下降沿处，CPU 对数据总线采样，从而获得数据。

（2）8086 存储器写周期时序（如图 2-44 所示）

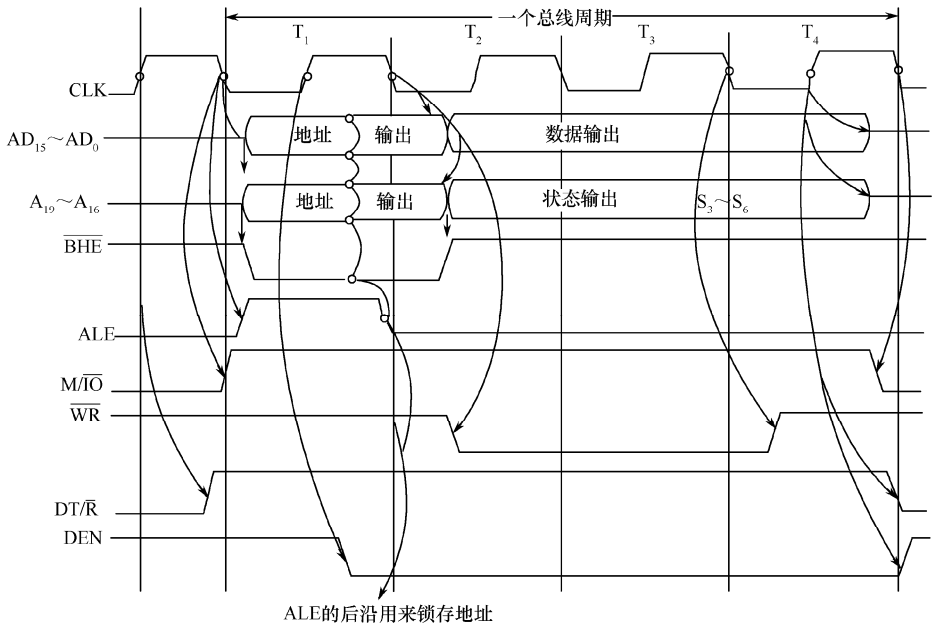


图 2-44　8086 存储器写周期时序

写总线周期和读总线周期的时序完全类似，唯一不同的是，在 T_2 前半周开始 $\overline{WR}$ 输出

電子工業出版社

低电平，在 T_3 结束才变为高电平，而 $\overline{RD}$ 则在 T_2 的后半周才变为低电平。$DT/\overline{R}$ 在整个写总线周期都输出高电平。

（3）8088 访问存储器时序

8086 与 8088 总线周期的时序非常相似，仅在地址/数据总线上有区别。8088 数据总线是 8 位，所以只有 AD_7～AD_0 是地址/数据复用线，而 A_{15}～A_8 是 8 条地址线。另外，8088 没有 $\overline{BHE}$ 信号。其时序图如图 2-45 所示。

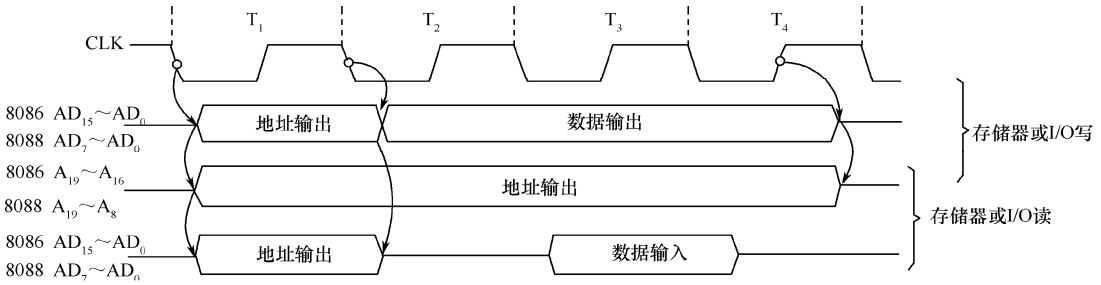


图 2-45　8088 与 8086 总线周期比较

（4）8086/8088 访问输入/输出端口的时序

8086/8088 访问外设的时序，即输入/输出时序，与 CPU 访问存储器的时序几乎完全相同，两者唯一区别是 $IO/\overline{M}$ （$M/\overline{IO}$）线，这里不再详述。

2.4.3　特殊总线操作时序

1．中断响应周期

图 2-46 是 8088 中断响应周期时序。当外部中断通过 CPU 的 INTR 引脚输入一个高电平时，表示外设向 CPU 提出中断请求（可屏蔽中断请求），当 CPU 允许中断（IF=1）时，CPU 在当前指令执行完后就响应中断，进入中断请求时序。中断响应时序中有两个连续的中断响应周期。在第一个中断响应周期，CPU 输出 $\overline{INTA}$ 负脉冲，表示 CPU 响应外设中断请求，用以撤销外设发出的 INTR 信号；在第二个中断响应周期，CPU 输出 $\overline{INTA}$ 负脉冲，通知外设向数据线上送 1 个字节的中断向量类型码，CPU 读入后，根据中断类型码自动在中断向量表中取该设备的中断服务程序的入口地址并转入中断服务程序。

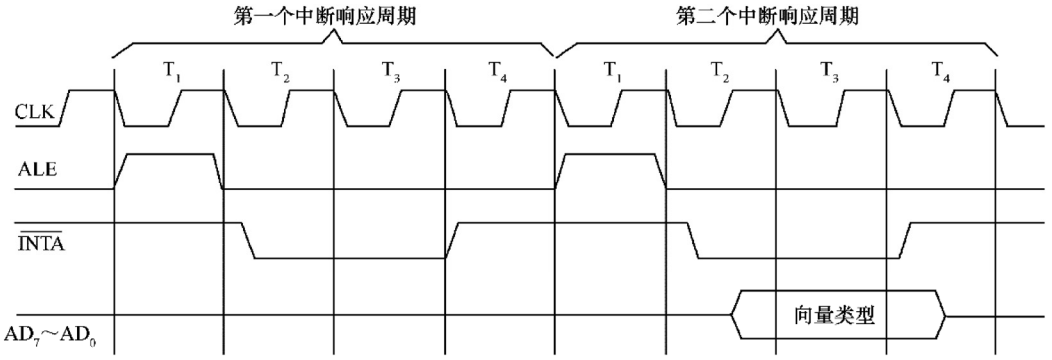


图 2-46　8088 中断响应周期时序

2．8086/8088 等待（WAIT）状态时序

8086/8088 插入等待状态 T_w 与 MN/$\overline{MX}$ 引脚（最小/最大模式）接 V_{CC} 或 GND 及正在执行的是否读/写周期无关。在任何总线周期，都可以在 T_2 和 T_4 之间插入 1～n 个等待时钟周期 T_w 来延长总线周期。

当存储慢速设备（存储器或 I/O 设备）数据时，必须插入等待状态来延长总线周期，这个任务由 READY（准备就绪）信号来实现。当被访问对象的数据传输速度与 CPU 存取数据的速度匹配时，READY 线处于高电平；只有当被访问对象的数据传输速度慢于 CPU 的存取速度时，READY 信号才在 T_2 结束的下降沿之前，变为低电平。CPU 在 T_3 的上升沿采样 READY 线。若 READY 为低电平，则自动插入一个 T_w 状态；若 READY 为高电平，则不插入 T_w 状态，并在 T_3 结束进入 T_4 状态。当插入 T_w 时，在每个 T_w 状态的上升沿继续采样 READY 线，若仍为低电平，则继续插入下一个 T_w，直到采样到高电平为止，才结束等待状态，进入 T_4 状态，结束总线周期，其时序如图 2-47 所示。

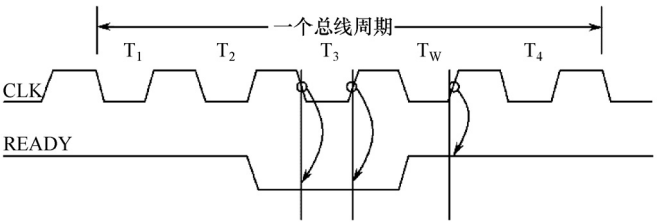


图 2-47　数据 READY 信号插入 T_w 状态示意

3．DMA 方式下的控制时序

DMA（Direct Memory Access）方式是 CPU 让出总线（悬浮状态），使外部设备和存储器之间（不通过 CPU）直接传送数据的方式。它通常使用在外部设备和存储器之间有大量数据需要传送及外部设备本身工作速度很快的情况下，尤其用于高速数据采集系统中。

CPU 在每个时钟周期的上升沿采样 HOLD 信号，如果允许让出总线，就在当前总线周期完成时（T_4 状态），向 HLDA 引脚发出一个回答信号，响应 HOLD 请求。同时，CPU 使地址/数据总线和有关控制信号线进入高阻状态，即放弃总线控制权。另一方面，总线请求部件（如 DMA 控制器 DMAC）收到有效 HLDA 信号后，就获得了总线控制权。在此期间，HOLD 和 HLDA 都保持高电平，在总线占有部件（当前总线主控）用完总线之后，将把 HOLD 信号变为低电平，表示放弃对总线的占用。CPU 收到低电平的 HOLD 信号后，将 HLDA 变为低电平，又获得了总线的控制权。

4．8086/8088 总线空闲周期

只有当 CPU 和存储器及 I/O 端口之间传输数据时，CPU 才执行总线周期。CPU 在不执行总线周期时，BIU 总线接口部件就不和总线打交道，此时进入总线空闲周期 T。总线空闲周期中，状态信号 S_6～S_3 和前一个总线周期（可能为读或写周期）是一样的。如果前一个总线周期是写周期，地址/数据复用引脚上还会在空闲周期中继续驱动前一个总线周期的数据 D_{15}～D_0。如前面一个总线周期是读周期，则 AD_{15}～AD_0 在空闲周期中处于高阻态。

在空闲周期中，尽管 CPU 对总线进行空操作，但在 CPU 内部，仍然进行着有效的操作。比如执行某个运算，在内寄存器之间传输数据等。实际上，这些动作都是在执行部件 EU 中进行的。总线空操作是总线接口部件 BIU 对执行部件 EU 的等待。

電子工業出版社

2.5 典型 CPU 应用系统

8086/8088 CPU 不包括存储器和 I/O 端口，也就是说，它们只能进行数据处理，不能记忆，更不能与外界交换信息。为此，8086/8088 CPU 芯片必须加上必要的支持芯片，如时钟电路、地址锁存器、总线驱动器以及存储器和 I/O 接口电路及基本外围设备，才能构成一台完整的微机系统。为了适用不同的应用场合，可以设计两种不同的工作模式：单 CPU（最小）系统和多 CPU（最大）系统模式。以 8086/8088 CPU 为核心，将微机各部件连接起来的一组线称为系统总线，如第 1 章的图 1-2 所示。本节以 8086/8088 CPU 为核心，说明如何通过系统总线来设计两种不同模式的系统。

2.5.1 8086/8088 支持芯片

1．8284 时钟发生器

8086/8088 CPU 内部没有时钟发生器，而 8284 时钟发生器是 Intel 公司专门为 8086/8088 系统配套设计的单片时钟发生器。它能为 CPU 提供时钟、准备就绪（READY）、复位（RESET）信号，还可向外提供晶体振荡信号（OSC）、外围芯片所需时钟 PCLK 等其他信号。

8284 时钟发生器的引脚及内部结构如图 2-48 所示，共 18 个引脚（PIN），采用双列直插封装（DIP）。下面简要介绍各引脚的功能。

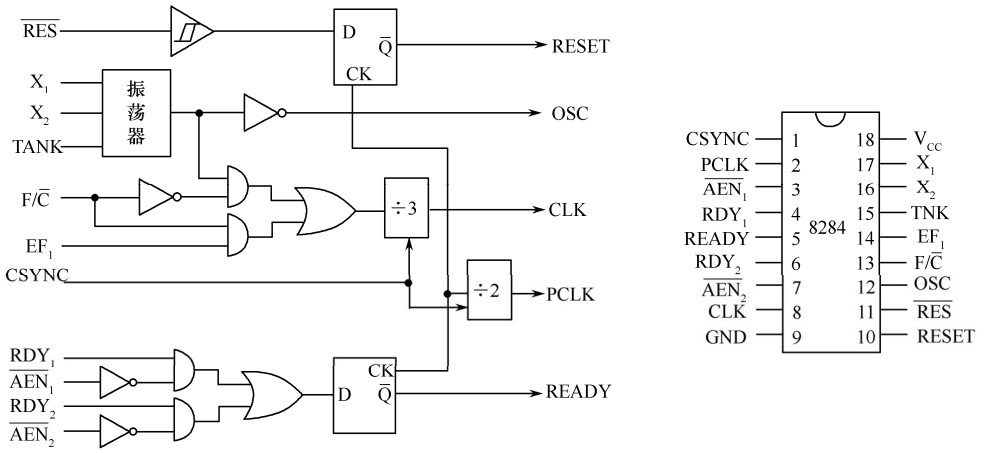


图 2-48　8284 时钟发生器内部结构及引脚

CLK：系统时钟，输出信号。直接接到 CPU 及其他以此为时钟的设备上，其频率为晶体频率或外接频率输入端 EF_1 信号频率的 1/3。CLK 信号占空比为 1/3。

$F/\overline{C}$：频率/晶振选择，输入信号。当 $F/\overline{C}$ =0（低电平）时，选择 CLK 为晶体所产生；当 $F/\overline{C}$ =1（高电平）时，CLK 由外接输入频率产生。

X_1、X_2：晶体输入，输入信号。X_1、X_2 直接接晶体的两个引脚，晶体的频率（14.318 MHz）为 CPU 所需时钟频率的 3 倍（4.77 MHz）。

PCLK：外部设备时钟，输出信号。输出波形占空比为 1/2，其频率（2.5 MHz）约为 CLK 的 1/2。

EF_1：外接输入频率，输入信号。当 $F/\overline{C}$ =1 时，选择 CLK 由该端输入信号产生。要求

電子工業出版社

输入信号的频率是 CLK 的 3 倍方波。

$\overline{AEN_1}$、$\overline{AEN_2}$：地址允许信号，输入信号。其状态分别控制 RDY_1 及 RDY_2。当 $\overline{AEN_1}$、$\overline{AEN_2}$ 为低时，RDY_1 和 RDY_2 产生 READY 信号，致使 CPU 产生等待周期。

RDY_1、RDY_2：总线就绪信号，输入信号。当系统总线上某个设备已收到数据或已准备好数据时，该设备可使 RDY_1 或 RDY_2 有效。

READY：就绪信号，输出信号。它由 RDY_1 或 RDY_2 形成。高电平表示已准备好；低电平时，使 CPU 产生等待周期。

$\overline{RES}$：外部复位信号，输入信号。经过施密特触发器及 D 触发器产生加到 CPU 的复位信号 RESET。

CSYNC：实现多片 8284 相位相同的时钟信号。输入信号。

一般情况下，8284 与 8086/8088 CPU 的连接如图 2-49 所示，未用到的引脚 OSC、EF_1 等可以空着不用。

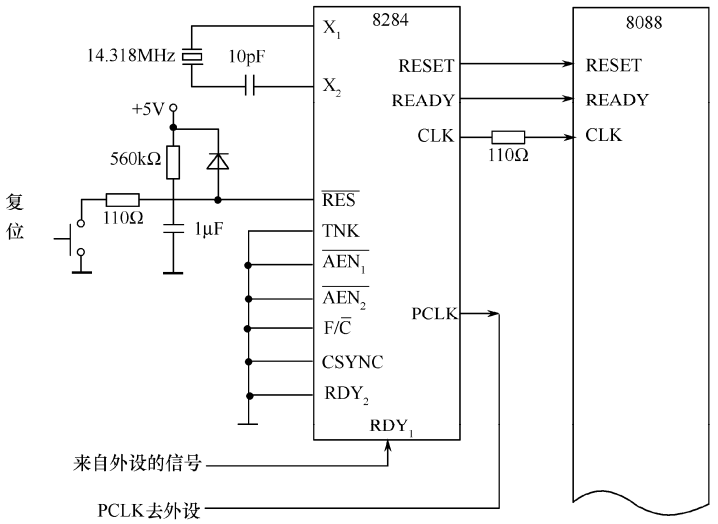


图 2-49　8284 与 8088 连接的一种方案

2．8282/8283 8 位三态输出锁存器

8086/8088 的 AD_{15}～AD_0/AD_7～AD_0 既可作为低 16 位/8 位地址线，又可作为 16 位/8 位数据线，为了把地址信息分离出来保存，为外接存储器或外设提供 16 位/8 位地址信息，一般需要外加三态锁存器，并由 CPU 产生的地址锁存允许信号 ALE 的下跳变将地址信息锁存入 8282/8283 锁存器中。

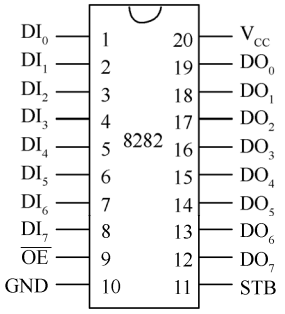
8282/8283 引脚及真值表如图 2-50 所示，采用 20 个引脚，双列直插封装。下面简要介绍 STB 及 $\overline{OE}$ 引脚的功能。

STB：选通信号，输入信号，高电平有效。当 STB 为高电平时，输出 DO_7～DO_0 随输入 DI_7～DI_0 而变，即起传输作用；当 STB 由高电平变到低电平时，将输入数据锁存。

$\overline{OE}$：输出允许，低电平有效。当 $\overline{OE}$ 为低电平时，将被锁存的信号输出；当 $\overline{OE}$ 为高电平时，8282/8283 输出呈高阻状态。在系统中，$\overline{OE}$ 一般接地，保证总是允许输出状态。

8282/8283 接入系统方式如图 2-51 所示。最后说明一点，8283 的功能与 8282 完全相同，仅仅是输入/输出反相而已。

電子工業出版社



8282真值表

$\overline{OE}$	STB	输出
1	X	高阻
0	1	$DO_i=DI_i$
0	↓	锁存

图 2-50　8282 三态锁存器及其真值表

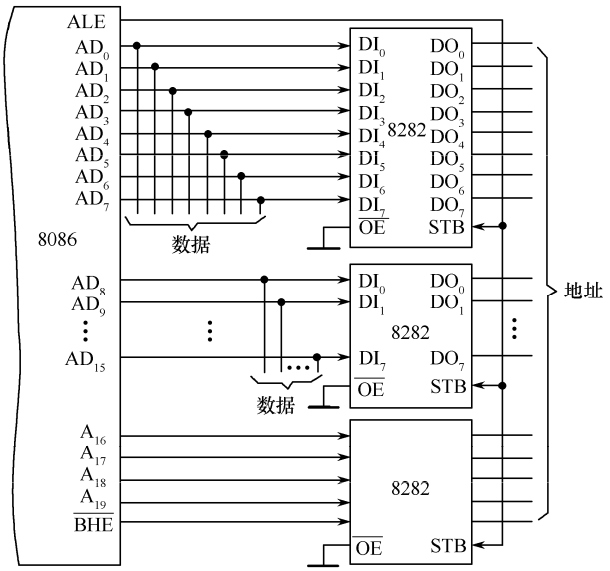


图 2-51　8282 锁存器和 8086 芯片的连接

3．8286/8287 并行双向总线驱动器

8 位并行双向总线驱动器用来将数据总线上数据接收到 CPU 或把 CPU 的数据发送到数据总线上去，另一方面用以增加数据总线的带负载能力。

8286/8287 引脚及一位内部结构如图 2-52 所示，采用 20 引脚，双列直插封装。下面简单介绍控制信号 $\overline{OE}$、T 的作用。

T：传输方向控制端，输入信号。当 T 为高电平时，数据由 $A_i \rightarrow B_i$（发送）；当 T 为低电平时，数据由 $B_i \rightarrow A_i$（接收）。

$\overline{OE}$：允许输出端，输入信号。当 $\overline{OE}$ 为低电平时，允许输出；当 $\overline{OE}$ 为高电平时，输出呈高阻状态。

8286/8287 接入系统方式如图 2-53 所示，$\overline{OE}$ 总是与 $\overline{DEN}$ 相接，T 则与 DT/$\overline{R}$ 相连。最后说明一点，8287 功能与 8286 完全相同，仅仅是输入/输出反相而已。

4．8288 总线控制器

8288 总线控制器专门为 8086/8088 CPU 构成多 CPU（最大模式）而设计，用以提供有关的总线命令和具有较强的驱动能力，使整个系统性能大大提高。

電子工業出版社

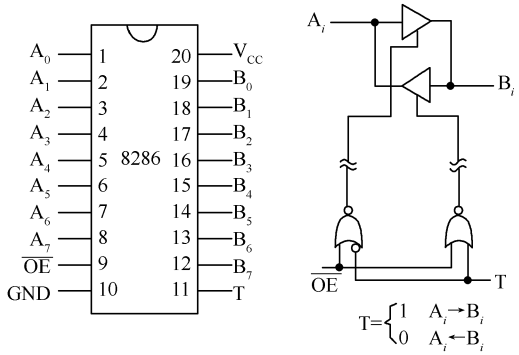


图 2-52　8286 芯片引脚及一位内部结构

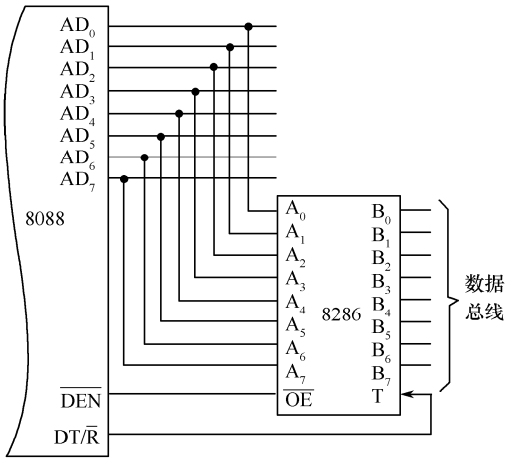


图 2-53　8286 收发器和 8088 芯片的连接

图 2-54 是 8288 引脚与 CPU 连接的两组输入信号和输出信号。下面简要介绍这两组信号。

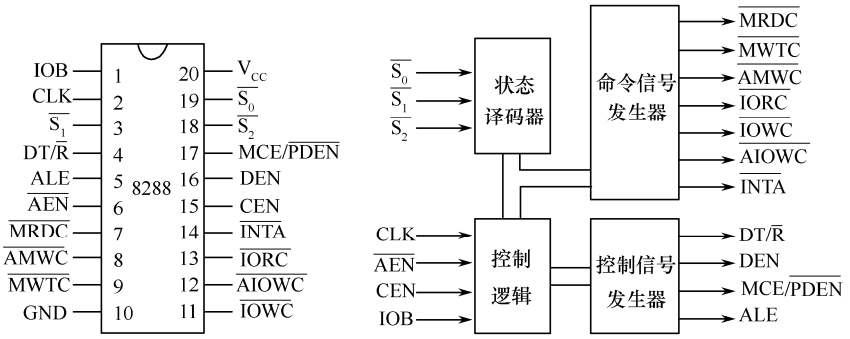


图 2-54　8288 总线控制器引脚结构示意图

$\overline{S}_2$、$\overline{S}_1$、$\overline{S}_0$：状态译码信号，输入信号。8288 通过对这些状态译码，产生相应总线命令信号和输出控制信号。

CLK：时钟信号，输入信号。通常接 8284 CLK 端。

$\overline{AEN}$：地址允许信号，输入信号。当 $\overline{AEN}$ 为低电平时，允许 8288 的各种命令输出；当 $\overline{AEN}$ 为高电平时，各命令处于高阻状态。

CEN：命令允许信号，输入信号。当 CEN 为低电平时，8288 所有命令信号及 $\overline{DEN}$、$\overline{PDEN}$ 全无效；当 CEN 为高电平时，允许上述信号输出。

IOB：总线方式输入控制信号，输入信号。当 IOB 为低电平时，8288 工作于系统总线方式；当 IOB 为高电平时，它工作于 I/O 总线方式。

$\overline{AIOWC}$：超前 I/O 写命令信号，输出信号。在总线周期中提早由 AIOWC 发出一个 I/O 写命令，以便较早地通知 I/O 设备执行的是写命令。

$\overline{AMWC}$：超前存储器写命令，输出信号。其功能与 $\overline{AIOWC}$ 相似，只是由 $\overline{AMWC}$ 发出。

$\overline{IOWC}$：I/O 写命令，输出信号。通知 I/O 外设取数据总线上的数据。

$\overline{IORC}$：I/O 读命令，输出信号。通知 I/O 外设把数据放在数据总线上。

$\overline{MWTC}$：存储器写命令，输出信号。通知所选存储单元接收数据总线上的数据。

$\overline{MRDC}$：存储器读命令，输出信号。通知所选中存储单元将数据放在数据总线上。

電子工業出版社

MCE/$\overline{\text{PDEN}}$：具有双功能，输出信号。当 IOB 为低电平时，MCE 高电平有效。在中断控制器级联工作时，MCE 用于从主中断控制器读出级联地址送到级联总线上。当 IOB 为高电平时，用于控制外设数据，控制 I/O 总线上的数据收发器，此时 $\overline{\text{PDEN}}$ 低电平有效。

$\overline{\text{INTA}}$、DT/$\overline{\text{R}}$、ALE 及 DEN 与 8086/8088 CPU 相应引脚类似，不再赘述。

2.5.2 8086/8088 单 CPU（最小模式）系统

1．8088 单 CPU 系统

在这种系统中，8088 CPU 引脚与总线连接关系遵循以下原则：

① 8088 CPU 的 MN/$\overline{\text{MX}}$ 引脚接高电平 V_{CC}。

② 8088 CPU 的 IO/$\overline{\text{M}}$、$\overline{\text{RD}}$、$\overline{\text{WR}}$、$\overline{\text{INTA}}$ 引脚直接接在存储器和 I/O 端口相应控制线上。

③ 地址线、地址/数据线接到地址锁存器上。因为 AD_7～AD_0 是地址/数据线复用引脚，在发数据前必须先将地址锁存起来。一般用锁存器 8282 或 74LS373。当 STB（或 G）由高电平变为低电平时，芯片将输入端的信息存入锁存器。$\overline{\text{OE}}$ 为输出控制信号，当它为低电平时，将锁存器内的信息送到输出端。为此只需将 CPU 的 ALE 接到 STB（或 G）上即可。

④ 数据线有两种接法：直接接到数据总线上，或者经过数据总线驱动器接到数据总线上。IBM PC/XT 采用后一种。这是因为数据总线上连接的设备比较多，要求它有足够的负载驱动能力。数据是双向传输的，需要采用双向总线驱动器。常用的 8 位双向总线驱动器有 8286/8287 或 74LS245。双向数据总线驱动器 A_7～A_0 引脚接 8088 CPU AD_7～AD_0，其 B_7～B_0 引脚接数据总线的 D_7～D_0 引脚上。8088 CPU 的 DT/$\overline{\text{R}}$ 接 8286 的 T 引脚（74LS245 是接 DIR 引脚），当 DT/$\overline{\text{R}}$ 为高电平时，数据从 8088 CPU 发送到系统总线上去；当 DT/$\overline{\text{R}}$ 为低电平时，CPU 则从系统数据总线上接收数据。8286 的 $\overline{\text{OE}}$ 引脚（74LS245 则为 $\overline{\text{G}}$）与 8088 CPU 的 $\overline{\text{DEN}}$ 引脚相接。8088 CPU 在 $\overline{\text{DEN}}$ 端为低电平期间，才允许数据输入/输出。

⑤ 8088 CPU 的 CLK 时钟，是由 8284 时钟发生器/驱动器提供的。8284 还为 8088 提供定时和宽度符合要求的 RESET 复位信号及符合同步要求的 READY“就绪”信号。

8088 单 CPU（最小模式）系统结构如图 2-55 所示。

2．8086 单 CPU 系统

图 2-56 是以 8086 为 CPU 的单 CPU（最小模式）系统结构。8086 的数据总线是 16 位的，所以需用两片 8286 双向数据驱动器。此外，8086 CPU 能直接寻址 1 MB 存储空间，可把它分为两个 512 KB 的存储体。其中一个奇存储体由奇地址单元组成，用于存储 16 位数据的高字节；另一个存储体由偶地址单元组成，用于存储 16 位数据的低字节。前者称为奇地址存储体，后者称为偶地址存储体。偶地址存储体的 8 位数据线，与 CPU 16 位数据总线的 D_7～D_0 连接；奇地址存储体的 8 位数据线与 CPU 16 位数据总线的 D_{15}～D_8 连接。地址总线 A_{19}～A_1 提供存储体片选的高位地址，A_0 和 $\overline{\text{BHE}}$ 区分奇偶地址存储体。当 A_0=0 时，选中偶地址存储体；当 A_0=1 时，不选中偶地址存储器。当 $\overline{\text{BHE}}$=0 时，选中奇地址存储体；当 A_0 和 $\overline{\text{BHE}}$ 两个信号都为低电平时，可同时对两个存储体进行读/写操作，即存储 16 位数据；而当 $\overline{\text{BHE}}$=0，A_0=1 时，只对奇地址存储体进行读/写操作；当 A_0=0，$\overline{\text{BHE}}$=1 时，只对偶地址存储体进行读/写操作。除此之外，与 8088 CPU 系统相同。

電子工業出版社

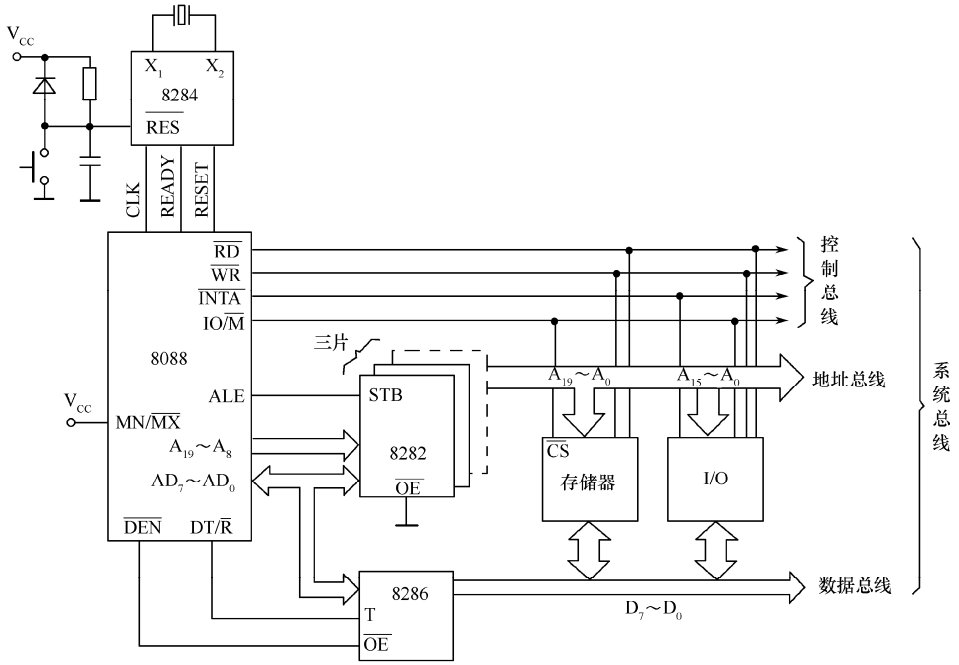


图 2-55　8088 单 CPU（最小模式）系统结构示意

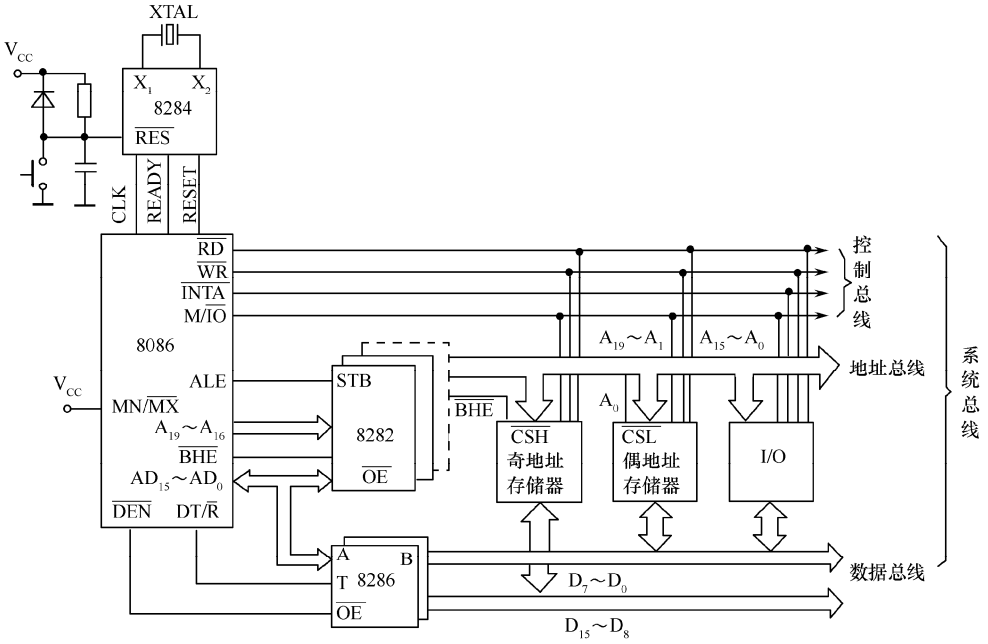


图 2-56　8086 单 CPU（最小模式）系统结构

2.5.3　8086/8088 多 CPU（最大模式）系统

多 CPU 模式又称为多处理器系统或最大模式系统，是在一个系统中存在两个或两个以上的 CPU。如在 IBM PC/XT 扩充系统中，一般加入 8087 浮点运算协处理器和 8089 I/O 处理器。一个多 CPU 系统，需要解决好以下 4 方面的问题：

電子工業出版社

- 多 CPU 并行处理时，各 CPU 之间同步的问题。
- 各 CPU 任务协调处理，并保证协调操作。
- 多 CPU 系统共用设备的共享和分配问题。
- 系统总线使用权的占有问题。

在实际的 IBM PC/XT 系统中，增加 8087 和 8089 辅助处理器后，并不是平等的并行操作，而是在主处理器 8088 的协调控制下进行操作。在任何情况下，两个处理器不能在某一时刻同时访问系统总线，各辅助处理器只有在请求主处理器并获得允许之后才能占用总线。各辅助处理器与主处理器之间采用异步通信方式交换数据。

根据以上原则构成的多 CPU 系统，在系统结构、总线时序和控制信号的定义及主处理器的引脚方面，都要做出相应变动。

1．以 8088 为核心构成的多 CPU 系统

图 2-57 是以 8088 为主的多 CPU 系统结构。

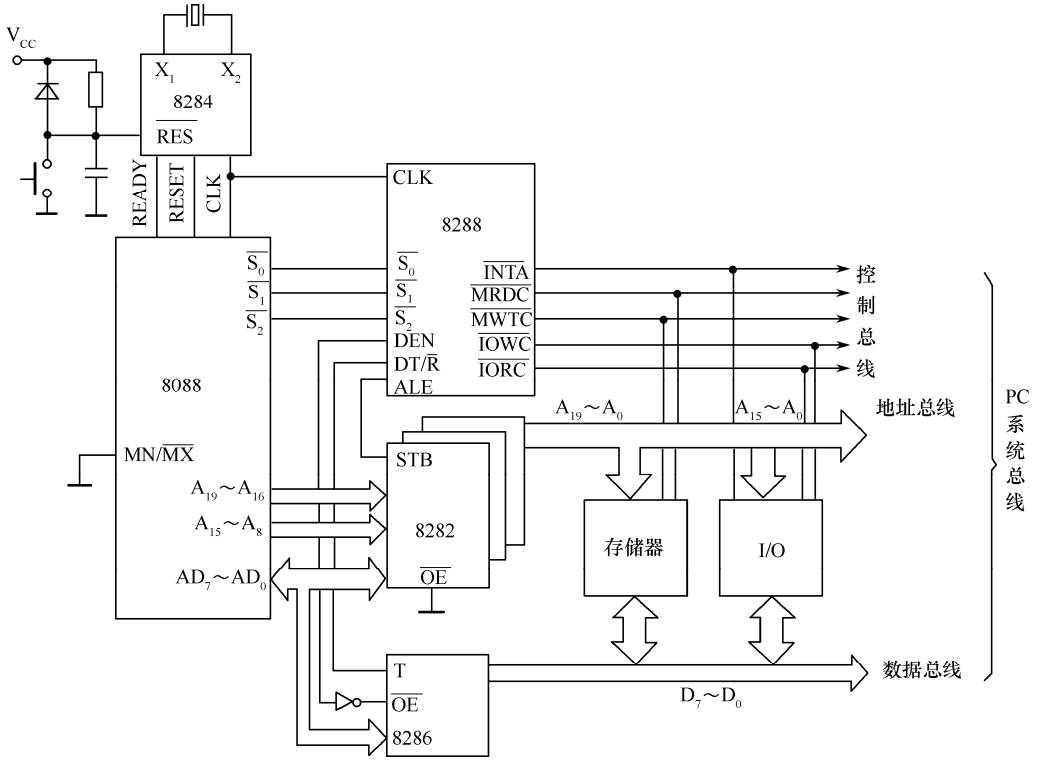


图 2-57　8088 多 CPU 系统结构

8088 用于多 CPU 系统时，MN/$\overline{MX}$ 引脚必须接地。由于 8088 不能直接提供总线控制信号，而必须用总线控制器 8288 对 CPU 提供的状态信息 $\overline{S}_2$、$\overline{S}_1$、$\overline{S}_0$ 进行译码来产生总线控制信号。为此，将几个 CPU 接到同一组系统总线上，或为了给 8088 提供几组系统总线，必须使用 8288 总线控制器。其主要作用是把原来的状态信号 $\overline{S}_2$、$\overline{S}_1$、$\overline{S}_0$ 及输入控制信号 CLK、$\overline{AEN}$、CEN、IOB 变成相应总线命令和控制信号。

8288 总线控制器主要功能如下。

① 状态信息译码功能，如表 2-13 所示。

電子工業出版社

表 2-13　8288 状态输入/输出命令对应关系

状态输入 $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$	CPU 总线操作周期	8288 产生命令	功　能
0　0　0	中断响应	$\overline{INTA}$	8288 输出的中断响应
0　0　1	I/O 读	$\overline{IORC}$	8288 I/O 读控制信号
0　1　0	I/O 写	$\overline{IOWC}$，$\overline{AIOWC}$	8288 I/O 写控制信号
0　1　1	停机	/	/
1　0　0	取指令周期	$\overline{MRDC}$	8288 存储器读控制信号
1　0　1	读存储器	$\overline{MRDC}$	8288 存储器读控制信号
1　1　0	写存储器	$\overline{MWTC}$，$\overline{AMWC}$	8288 存储器写，提前写
1　1　1	空操作	/	/

② 能产生系统总线的控制信号或访问 I/O 总线控制信号（$\overline{INTA}$、$\overline{IORC}$、$\overline{IOWC}$ 等）。

③ 可使总线控制信号浮空，允许 DMA 操作。

④ 提供专门为慢速存储器或 I/O 设备设计的预写控制信号 $\overline{AIOWC}$、$\overline{AMWC}$。

⑤ 可产生地址锁存器的锁存允许信号 ALE 和双向数据驱动器的控制信号 DEN、$DT/\overline{R}$。

⑥ 可产生简单或级联中断逻辑的控制信号 $MCE/\overline{PDEN}$。当 IOB 接低电平时，$MCE/\overline{PDEN}$ 引脚输出 MCE 信号，在中断响应总线周期的 T_1 状态有效，用于将中断控制器 8259A 的级联地址送至地址总线的同步信号。当 IOB 接高电平时，$MCE/\overline{PDEN}$ 引脚输出 PDEN 信号，低电平有效，并与 $\overline{DEN}$ 信号的功能和时序相同，但相位相反，它作为响应 I/O 总线收发器的允许信号。

2．以 8086 为核心构成的多 CPU 系统

图 2-58 为 8086 多 CPU 系统结构，其基本系统构成与图 2-57 类似，不再详述。

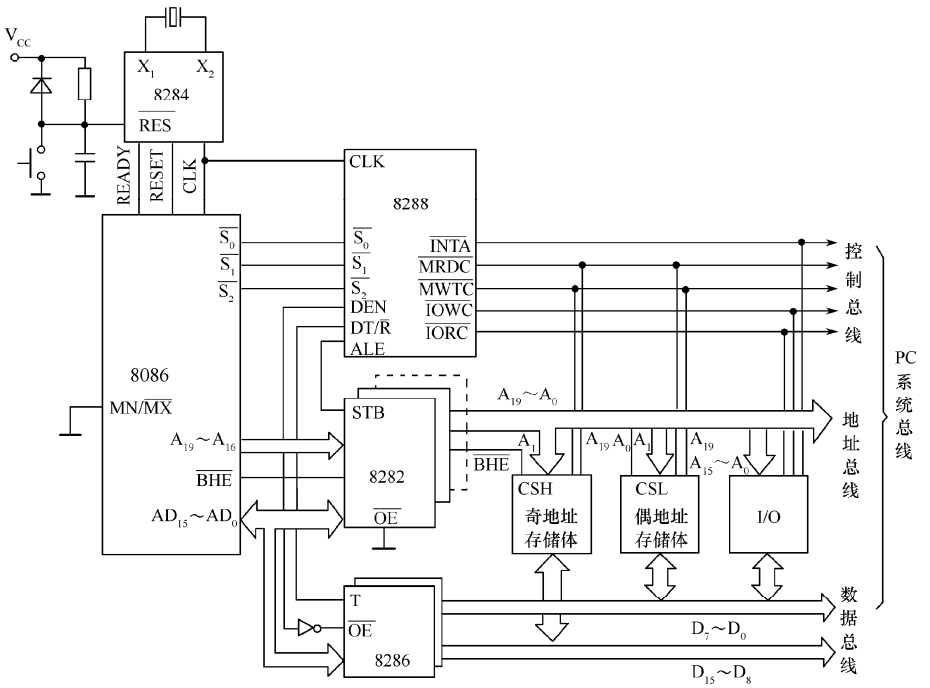


图 2-58　8086 多 CPU 系统结构

電子工業出版社

2.6 CPU 的工作模式

80x86 CPU 共有 4 种工作模式：实地址模式、保护模式、虚拟 8086 模式和系统管理模式。其中，8086/8088 CPU 只能工作于实地址模式，80286～80486 CPU 可以工作于实地址模式、保护模式和虚拟 8086 模式，Pentium 及其以上 CPU 可以工作在全部 4 种模式下。下面分别介绍 4 种模式的特点并给出其相互切换的控制方式。

2.6.1 实地址模式

实地址模式（Real Mode）是最基本的工作方式，它实际就是 16 位 8086/8088 CPU 的工作模式，其后的 CPU 为了与其兼容，都支持这种工作方式。在该模式下，原有 16 位 CPU 的程序不加任何修改就可以在 80286 及其以上的 CPU 上运行。

在实地址模式下，CPU 的地址线中只有低 20 位起作用，即能寻址的物理存储空间为 1 MB，其存储器的管理方式与 8086/8088 CPU 存储器的管理方式相同。系统复位时，CR_0 的 PE 位自动清 0，进入实地址方式。MS-DOS 操作系统要求 CPU 工作于实地址模式。

2.6.2 保护模式

通常在程序运行过程中，需要防止以下情况的发生：① 应用程序破坏了系统程序；② 某一应用程序破坏了其他应用程序；③ 错误地把数据当成程序来运行。为了避免以上情况的发生，所采取的措施称为“保护”。

保护模式（Protected Mode）是一种建立在虚拟存储器和保护机制基础上的工作模式，可最大限度地发挥 CPU 所具有的存储管理功能及硬件支持的保护机制，为多任务、多用户操作系统的实现提供硬件支持。

在保护模式下，CPU 通过设立 0～3 级 4 个特权级，实现应用程序之间及应用程序与操作系统之间的有效隔离，其保护结构如图 2-59 所示。0 级为最高特权级，分配给操作系统内核；操作系统的系统服务程序为 1 级；应用系统服务程序为 2 级；3 级为最低特权级，分配给应用程序。每个存储段都与同一个特权级关联，当前活跃的代码段的特权级为 CPL（Current Privilege Level）。CPU 的保护机制规定，对给定 CPL 的执行程序，只允许访问同一级或低特权级的数据段，否则将产生异常。

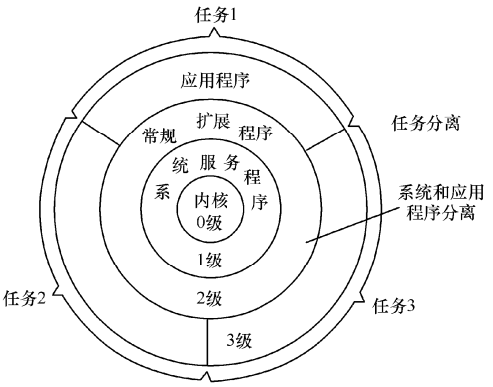


图 2-59 4 级特权级保护结构示意

電子工業出版社

保护模式的保护机制主要指两方面：① 通过给每个任务分配不同的地址空间，使任务之间完全隔离；② 任务内部的保护机制，以保护操作系统存储空间及特别的 CPU 寄存器，使其不被其他应用程序所破坏。

保护模式下的虚拟地址空间被分为全局地址空间和局部地址空间。每个任务各自占有的虚拟地址空间为局部地址空间，其内部的代码和数据是任务私有的，需要与系统中的其他任务相隔离。各任务共用的一部分虚拟地址空间为全局地址空间，操作系统存储在全局地址空间中，可以被所有任务共享，而且任务对它的访问不会破坏它的内容。

保护模式分两步实现虚拟地址空间到物理地址空间的映射，也就是分两步实现地址转换，但第二步是可选的。通过描述符表和描述符、分段管理机制实现虚拟地址空间到线性地址空间的映射，实现把二维虚拟地址转换为一维线性地址，这一步总是存在的。

分页管理机制把线性地址空间和物理地址空间分别划分为大小相同的块，这样的块称为页（Page）。通过在线性地址空间的页与物理地址空间的页之间建立映射表，分页管理机制实现线性地址空间到物理地址空间的映射，实现线性地址到物理地址的转换。分页管理机制是可选的，在不采用分页管理机制时，线性地址空间就等同于物理地址空间，线性地址就等同于物理地址。

2.6.3 虚拟 8086 模式

虚拟 8086 模式（Virtual 8086 Mode）是一种特殊的保护模式，工作方式与保护模式相同，但其对逻辑地址的解释是按照 8086/8088 的方式进行的。在这种模式下，不仅可以执行 8086/8088 程序，而且支持多个 8086/8088 实模式程序的运行，其实质是为运行于 32 位 CPU 上的 8086 程序提供独立的虚拟机。

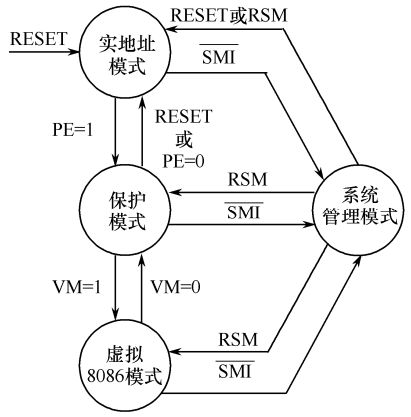


图 2-60 CPU 4 种工作模式的相互转换

2.6.4 系统管理模式

系统管理模式（System Management Mode，SMM）是 Pentium CPU 新增加的工作模式，它不是作为应用程序或系统级特性，而是为实现高级的系统功能，如电源管理和安全性。进入系统管理模式，CPU 的状态被保存，并有其特定的存储空间，这时系统可以进入休眠状态，其后在有键按下或鼠标移动时能自动唤醒，并使系统从原先的休眠中断点开始继续工作。

上述 4 种模式之间的转换关系如图 2-60 所示，其中 PE 为保护模式使能信号，VM 为虚拟 8086 模式有效信号，$\overline{\text{SMI}}$ 表示系统管理中断信号有效，RSM 表示系统管理方式返回指令。

习 题 2

2-1 了解 ARM 的各系列处理器特性，并给出列表的说明。

2-2 除了 2.1 节给出的几种微处理器，还有哪些目前应用较为广泛的处理器？请给出简单

電子工業出版社

的说明。

2-3　通过查阅资料，分析多核微处理器的发展现状及趋势，并对发展中的核心技术或制约点加以说明。

2-4　8086/8088 为什么要分为 EU 和 BIU 两部分？每一部分由几部分组成？

2-5　简述 8086/8088 指令队列的作用及其工作过程。

2-6　试述 SP、IP、BP、SI、DI 寄存器名称的由来。它们能否作为通用寄存器？

2-7　8086/8088 有几位状态位？有几位控制位？其含义各是什么？

2-8　8086/8088 CPU 使用的存储器为什么要分段？怎样分段？

2-9　8086 与 8088 CPU 主要区别有哪些？

2-10　什么是逻辑地址？什么是物理地址？它们之间有什么联系？各用在何处？

2-11　什么是基地址？什么是偏移量？它们之间有何联系？

2-12　若 CS 为 A000H，试说明当前代码段可寻址的存储空间的范围。

2-13　设当前数据段位于存储器 B0000H～BFFFFH 存储单元，DS 段寄存器内容为多少？

2-14　设双字 12345678H 的起始地址是 A001H，试说明这个双字在存储器中如何存放？

2-15　已知堆栈段寄存器 SS=A000H，堆栈指示器 SP=0100H，试将数据 1234ABCDH 推入堆栈，画出进栈示意图，最后栈顶 SP=？

2-16　试求出下列运算后的各状态标志，并说明进位标志和溢出标志的区别。

（1）1278H+3469H　　（2）54E3H−27A0H

2-17　8088 CPU 工作在最小模式（单 CPU）和最大模式（多 CPU）的主要特点是什么？它们有何区别？

2-18　在 8088 CPU 工作在最小模式时：

（1）当 CPU 访问存储器时，要利用哪些信号？

（2）当 CPU 访问 I/O 端口时，要利用哪些信号？

（3）当 HOLD 有效并得到响应时，CPU 的哪些信号置高阻？

2-19　简述时钟周期、机器周期、总线周期和指令周期的含义及其区别。

2-20　为什么要了解 8086/8088 CPU 时序？

2-21　试绘出 8086 单 CPU 系统访问 I/O 端口总线周期的时序图。

2-22　什么情况下插入 T_W 等待周期？插入多少 T_W，取决于什么因素？

2-23　什么情况会出现总线空闲周期？

2-24　试用表格列出 8088 单 CPU 方式下读/写存储器状态时，IO / $\overline{M}$、DT / $\overline{R}$、$\overline{DEN}$、$\overline{RD}$、$\overline{WR}$ 的值。如读存储器 $\overline{RD}$ =0。

2-25　8086/8088 单 CPU 和多 CPU 系统各有什么主要特点？差别在哪里？

2-26　8086 构成系统时可分为哪两个存储体？它们如何与地址、数据总线连接？

2-27　简述 80386 和 Pentium CPU 基本组成与各部分作用。

2-28　什么是实地址模式？保护模式？虚拟 8086 模式？三种模式的主要特点是什么？

2-29　试说明 80386 和 Pentium CPU 控制寄存器的作用。

2-30　CS 寄存器在保护模式下称为什么？DS 寄存器呢？

2-31　选择符中三个域的名字和大小各是多少？

2-32　如果 GDT 寄存器的值为 0013000000FFH，装入 LDTR 的选择符为 0040H，那么将要装入缓存 LDT 描述符的起始地址是多少？

電子工業出版社

2-33　如果标志寄存器的 IOPL 位为 10，那么 I/O 指令的特权级为多少?

2-34　Pentium CPU 的物理地址空间和虚拟地址空间分别为多少？它们是如何计算出来的？

2-35　页转换产生的线性地址的三部分各是什么?

2-36　段描述符与什么寄存器相关联？系统段描述符呢？

2-37　选择符 0224H 装入了数据段寄存器，该值指向局部描述符表中从地址 00100220H 开始的段描述符。如果该描述符的字为

[00100220H]=0110H，　　[00100222H]=0000H

[00100224H]=1A20H，　　[00100226H]=0000H

那么“限长”和“基址”各为多少?

2-38　试说明 Pentium、Pentium Pro、Pentium Ⅱ、Pentium Ⅲ、P4 等 CPU 的基本特点。

電子工業出版社

第 3 章　微处理器指令系统

指令系统是微处理器所能执行指令的集合，这组指令集定义了计算机硬件所能完成的基本操作。不同系列的微处理器由于其内部结构不同而有不同的指令系统，随着处理器的升级指令系统也会不断扩充。CISC 处理器的指令系统比较丰富完善，且有专用指令实现特定的功能。RISC 处理器指令系统的特色是常用功能的指令简单高效，处理特殊任务则要通过组合指令来完成。

3.1　指令格式

指令格式体现了指令系统的设计思想。要掌握指令系统，先了解指令格式是必要的。

各种微处理器的指令格式都是类似的，主要由操作码和操作数两部分组成。操作码指示CPU执行哪种操作，是数据传送、加减乘除，还是逻辑操作等，是指令中不可缺少的部分。操作数指示指令执行过程中所需要的数据，如加法指令中的加数被加数等。指令操作数的给出方式比较复杂，可以是参加操作的数本身（立即操作数），也可以是存放操作数的寄存器（寄存器操作数）或存储单元地址（存储器操作数）。如何寻找（规定）操作数是寻址方式问题，将在3.2节介绍。微处理器汇编语言指令的一般格式如下：

```
        指令助记符        操作数列表        ；注释
```

指令助记符是指令的名称，对应机器指令中的操作码部分。操作数可以有多个，因指令不同而不同，按序排列并用“,”分开。“;”后是注释部分，不影响指令的执行。

80x86/Pentium CPU汇编语言指令的一般格式如图3-1所示，占1～16字节。一条指令在被汇编翻译成机器指令后，对应若干字节的机器码。

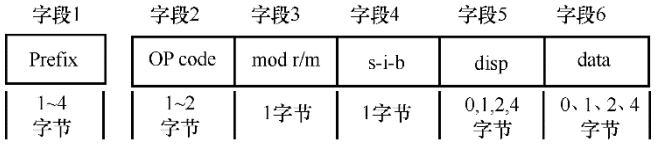


图 3-1　80x86/Pentium 指令的一般格式

在一条指令的 6 个字段中：2～6 为基本字段，1 是附加字段，简明意义如下。

（1）OP code 操作码字段。它规定指令的操作类型，说明指令所要完成的操作，同时指出操作数类型（字节/字/双字）、操作数传送方向、寄存器编码或操作数的符号扩展等。

（2）mod r/m 和 s-i-b 寻址方式字段。该字段规定寄存器/存储器操作数的寻址方式。mod r/m 为主寻址字节，规定操作数存放的位置（r/m）、存储器操作数有效地址 EA 的计算方法等。s-i-b 为“比例-变址-基址”寻址字节。一般访问存储器的指令中都含有主寻址字节，而“比例-变址-基址”寻址字节是否需要由主寻址字节的编码决定。

（3）disp 位移量字段。位移量是存储器操作数段内偏移地址的一部分。该字段指出位移量的大小，其长度为 0、1、2、4 字节。

電子工業出版社

（4）data 立即数字段。该字段指明立即操作数的数值大小，其长度也是 0、1、2、4 字节。8 位立即数与 16/32 位操作数一起使用时，汇编程序将它扩展至符号相同的 16/32 位数。同理，也可将 16 位立即数扩展至 32 位。

（5）Prefix 前缀字段。用于修改指令操作的某些属性。常用前缀有 5 类，每个前缀的编码为 1 字节。在一条指令前可同时使用多个指令前缀，不同前缀的前后顺序无关紧要。

① 段超越前缀：用于将前缀中指明的段寄存器取代指令中默认的段寄存器。

② 操作数宽度前缀：用于改变当前操作数宽度的默认值。

③ 地址宽度前缀：用于改变当前地址宽度的默认值。

④ 重复前缀：用于重复串的基本操作，以提高 CPU 处理串数据的速度。

⑤ 总线锁定前缀 LOCK：用于产生 LOCK 信号，以防止其他总线主控设备中断 CPU 在总线上的传输操作。

在指令中附加操作数宽度前缀或地址宽度前缀，可以保证 80386 及其以后的 CPU 在实模式和保护模式下均可执行 16 位和 32 位指令，实现 80x86/Pentium 系列 CPU 的兼容性。

在机器指令的 6 个字段中，只有操作码字段是必要的，其他字段是否需要，具体取决于所涉及的特定操作。

RISC 处理器，如 MIPS 或 ARM，指令格式的最大特征是机器码位数固定、编码紧凑。

MIPS处理器的汇编语言指令格式同80x86/Pentium，指令集采用32位编码，其3种机器码格式如图3-2所示，分别是寄存器类（R型）、立即数类（I型）和转移类（J型）。其中，SA（Special Area）字段比较复杂，可以由特定的CPU自定义，用来实现指令集的扩充。

	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
R 类型	操作码 OP	RS1	RS2	RD	SA	辅助操作码 OPX
I 类型	操作码 OP	源操作数寄存器 RS	目标操作数寄存器 RD	立即数 Immediate（16 bit）		
J 类型	操作码 OP	直接转移指令的目标地址 Target（26 bit）				

图 3-2　MIPS 的指令编码格式

ARM 体系结构从 ARM v1 发展到 ARM v8，指令集包含有 32 位和 16 位（Thumb）指令两类，指令功能不断扩大并向下兼容，ARM v8 还扩展了对 64 位寄存器的操作。ARM 汇编语言指令的一般格式如下：

```
指令助记符 条件 S     操作数列表          ; 注释
```

与 80x86/Pentium 指令格式略有不同，附着于助记符后的条件为指令执行的条件域，如 EQ、NE 等，是可选项；S 是指令的执行结果是否影响程序状态寄存器的后缀，有则影响。ARM 32 位或 16 位指令的机器码一般格式如图 3-3 和图 3-4 所示。

4 bit	3～4 bit	4～5 bit	4 bit	4 bit	11 bit
条件码	指令类型	操作码、寻址方式、操作是否影响 CPSR 值	源操作数寄存器	目标操作数寄存器	第二个源操作数、寄存器列表

图 3-2　ARM 32 位指令的一般格式

3～6 bit	1～3 bit	3～5 bit	3～8 bit	3 bit
指令类型	操作码	目标操作数寄存器、源操作数寄存器、立即数	源操作数寄存器、立即数、寄存器列表	第二个目标操作数寄存器

图 3-3　ARM Thumb 指令的一般格式

電子工業出版社

3.2 寻址方式

3.2.1 寻址方式与有效地址 EA 的概念

寻址方式就是指令中用来说明操作数怎样存放以及如何寻找操作数的方式。CPU 据此可以方便地访问各类操作数。通常，指令中的操作数有 3 种可能的存放位置：

- 操作数包含在指令中，即指令的操作数部分就是操作数本身。这种操作数称为立即数，对应的指令寻址方式称为立即寻址。
- 操作数存放在 CPU 的某个内部寄存器中，称为寄存器操作数。这时指令的操作数部分是 CPU 内部寄存器的编码，对应的寻址方式称为寄存器寻址。
- 操作数在内存的数据区中，称为存储器操作数。这时指令的操作数部分给出操作数的存储地址，指令执行时需要根据该地址找到需要的操作数。这种寻址方式称为存储器寻址。

CPU 指令中给出的存储器地址都是逻辑地址，逻辑地址与实际物理地址之间的映射由内存管理单元 MMU 来完成。以 Intel 80x86/Pentium 系列 CPU 指令为例，若内存中某一单元的逻辑地址用 DS:TABLE 来表示。在实地址模式下，DS 中是段基地址的高 16 位，称为段地址；对于保护模式，DS 中存放的是指向段描述符的 16 位段选择符，通过它可以得到 32 位的段基地址。TABLE 为段内偏移量，有 16 位（实模式下）和 32 位（保护模式下）两种。存储器寻址时，指令的操作数部分给出的地址就是这个段内偏移量。为了在程序设计时能够适应各种数据结构的需要，80x86/Pentium 系列 CPU 规定这个段内偏移量可以由几个基本部分组合而成，所以也把它称为有效地址 EA。

组成有效地址 EA 的基本部分包括：基址寄存器内容，变址寄存器内容，位移量，比例因子。这四个基本部分称为有效地址四分量。其中，基址、变址寄存器中通常为某局部存储区的起点；位移量是指令中的 disp 字段，是一个具体数值，也可以是符号常量或变量；比例因子是 32 位寻址方式中特有的部分。它们的组合情况和计算方法如下：

EA=(基址寄存器)+(变址寄存器×比例因子)+位移量

对于 16 位寻址和 32 位寻址方式，有效地址 4 种分量的使用规则有所不同，如表 3-1 所示。由这 4 种分量可组合出多种存储器寻址方式。

表 3-1 有效地址 4 种分量的使用规则

有效地址分量	16 位寻址	32 位寻址
基址寄存器	BX，BP	任何 32 位通用寄存器（EAX/EBX/ECX/EDX/ESI/EDI/EBP/ESP）
变址寄存器	SI，DI	除 ESP 外的任何 32 位通用寄存器（EAX/EBX/ECX/EDX/ESI/EDI/EBP）
比例因子	1	1，2，4，8
位移量	0，8，16（位）	0，8，32（位）

3.2.2 80x86/Pentium 各种寻址方式

80x86/Pentium CPU 指令系统的寻址方式有 11 种，对 8086/8088/80286（16 位寻址）来说只有 8 种，80386/80486 及 Pentium 系列 CPU 除了可 16 位寻址，还支持 32 位寻址方式。以下介绍中均以指令的源操作数为对象进行分析。

電子工業出版社

1．立即寻址

立即寻址方式下，操作数作为立即数直接包含在指令中，紧跟在操作码之后与其一起存放在代码段区域。因此，立即数总是和操作码一起被存入 CPU 的指令队列，在指令执行时不需再访问存储器。

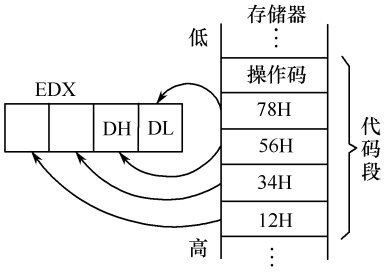


图 3-2　立即寻址方式

立即操作数可以是 8 位、16 位或 32 位。若是 16 位，则低位字节存放在相邻 2 字节存储单元的低地址单元中；若是 32 位，则低位字存放在相邻 2 字存储单元的低地址单元中。立即寻址方式仅用于源操作数，常用来给寄存器赋初值。例如：

```
MOV     BL，12H
MOV     AX，1234H
MOV     EDX，12345678H
```

其中，MOV　EDX, 12345678H 的执行过程如图 3-2 所示。

2．寄存器寻址方式

寄存器寻址方式的操作数存放在指令规定的 8 位、16 位或 32 位 CPU 内部寄存器中。例如：

```
INC     CL                  ; CL←CL+1
MOV     DS，AX              ; DS←AX
MOV     ECX，EAX            ; ECX←EAX
```

显然，这是 CPU 内部操作，不需要使用访问总线周期，所以指令的执行速度较快。

3．存储器寻址方式

存储器寻址时，操作数是在存储区中，指令的操作数部分指出此操作数的有效地址 EA，而段地址在默认的或段超越前缀指定的段寄存器中。根据 EA 的不同组合方式，存储器寻址方式又可分为以下 9 种。

（1）直接寻址

直接寻址是一种最简单的存储器寻址方式。在这种寻址方式下，指令中操作数部分直接给出操作数的有效地址 EA，它是 16 位或 32 位的位移量数据，与操作码一起放在代码段中。操作数一般在数据段（DS）中，这是一种默认方式。如果要对除 DS 段之外的其他段（CS、ES、SS、FS、GS）中的数据寻址，应在指令中增加段超越前缀指出段寄存器名。例如：

```
MOV     AX，[2000H]         ; 将 DS 段中 2000H 和 2001H 单元内容分别送 AL 和 AH
MOV     AX，ES:[2000H]      ; 将 ES 段中 2000H 和 2001H 单元内容分别送 AL 和 AH
```

汇编语言程序中，直接寻址中的有效地址 EA 也可用变量名形式给出。例如：

```
VALUE   DB   12H            ; 定义变量 VALUE 并赋初值
MOV     AL，VALUE           ; 将变量 VALUE 所指的字节单元内容 12H 存入 AL
```

图 3-3 是直接寻址方式的示意。

（2）寄存器间接寻址

操作数在存储区内，操作数的有效地址 EA 在指令指定的寄存器中，即 EA=（寄存器）。寄存器的使用规定在 16 位寻址和 32 位寻址时不一样。

① 16 位寻址时，EA 放在 SI、DI、BP、BX 中。这时又有两种段的默认情况。

電子工業出版社

若以 SI、DI、BX 间接寻址，则默认操作数在数据段中。例如：

```
MOV     AX, [SI]      ; 默认 DS 为段地址
```

若以 BP 间接寻址，则默认操作数在堆栈段中，提供了一种随机访问堆栈数据的方法。例如：

```
MOV     AX, [BP]      ; 默认 SS 为段地址
```

若操作数不在上述规定的默认段，则必须在指令的相应操作数前加上段超越前缀。例如：

```
MOV     CX, DS:[BP]
; “DS:”是该指令的段超越前缀
```

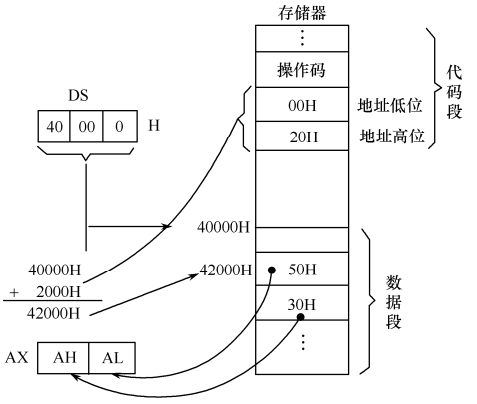


图 3-3　直接寻址方式

② 32 位寻址时，8 个 32 位通用寄存器均可作为寄存器间接寻址。例如：

```
MOV     CH, [EAX]
MOV     DX, [EBP]
```

这时，除了 EBP、ESP 默认段寄存器为 SS，其余 6 个通用寄存器均默认段寄存器为 DS。同样可以采用加段超越前缀的方法对其他段进行寻址。

由此可见，直接寻址中有效地址 EA 来自指令自身，相当于一个常量。寄存器间接寻址中有效地址 EA 来自寄存器，寄存器的内容由前面的指令确定，相当于一个变量。

（3）基址寻址

在这种寻址方式下，操作数的有效地址 EA=(基址寄存器)+位移量。式中，位移量在指令中给出并与操作码一起存放在代码段中。

① 16 位寻址时，BP 和 BX 作为基址寄存器。默认情况下，BX 以 DS 作为段寄存器，BP 以 SS 作为段寄存器，可以段超越。位移量是 8 位或 16 位。

② 32 位寻址时，8 个 32 位通用寄存器均可作为基址寄存器。其中 ESP、EBP 默认段寄存器为 SS，其余 6 个寄存器均默认段寄存器为 DS，可以段超越。位移量是 8 位或 32 位。例如：

```
MOV     AX, [BX+24]       ; 也可写成 MOV  AX, 24[BX]
MOV     ECX, [EBP+50]     ; 也可写成 MOV  ECX, 50[EBP]
MOV     DX, [EAX+TABLE]   ; 或写成 MOV  DX, TABLE[EAX]，TABLE 是符号常量或变量
```

（4）变址寻址

变址寻址方式下，有效地址 EA=(变址寄存器)+位移量。它的指令格式及寻址过程与基址寻址相同，区别仅在于将基址寄存器改成变址寄存器。

① 16 位寻址时，SI 和 DI 作为变址寄存器，且默认 DS 作为段寄存器。例如：

```
MOV     AX, COUNT[SI]
```

② 32 位寻址时，除 ESP 外的任何 32 位通用寄存器均可作为变址寄存器，且 EBP 以 SS 为默认段寄存器，其余以 DS 为默认段寄存器。例如：

```
MOV     EAX, 5[EBP]
MOV     ECX, DATA[EAX]
```

基址或变址寻址非常适合对一维数组元素进行检索操作，常用位移量表示数组起始地址，基址或变址寄存器表示数组元素的可变下标，通过修改基址或变址寄存器的内容可方便地访问数组中的任意元素。

電子工業出版社

（5）比例变址寻址

比例变址寻址只适合 32 位寻址的情况。有效地址 EA=(变址寄存器)×比例因子+位移量，乘以比例因子的操作是在 CPU 内部靠硬件完成的。例如：

```
MOV     EAX, TABLE[ESI*4]
```

比例变址寻址的作用和基址或变址寻址的作用相似，但更适合对多字节一维数组元素进行检索，当数组元素大小为 2、4、8 字节时，可取比例因子 2、4、8，访问更方便、高效。

（6）基址加变址寻址

基址加变址寻址方式中，操作数的有效地址 EA=(基址寄存器)+(变址寄存器)。它也有 16 位寻址和 32 位寻址两种情况。每种情况下基址、变址寄存器的使用规定和段寄存器的默认规定与前面所述相同。注意，当一种寻址方式中基址、变址寄存器默认的段寄存器不同时，一般由基址寄存器来决定默认哪一个段寄存器作为段地址。若在指令中规定了段超越，则可用其他段寄存器作为段地址。例如：

```
MOV     AX, [BX+SI]       ; 或写成 MOV  AX, [BX][SI]，BX 决定默认 DS 为段寄存器
MOV     EAX, [EDX][EBP]   ; 由 EBP 决定默认 SS 为段寄存器
```

基址加变址寻址适合检索二维数组元素和双重循环等操作。

（7）基址加比例变址寻址

这种寻址方式只适合 32 位寻址的情况。有效地址 EA=(基址寄存器)+(变址寄存器)×比例因子。例如：

```
MOV     ECX, [EDX*4][EAX]         ; 或  MOV    ECX, [EDX*4+EAX]
MOV     AX, [EBX*8][ESI]          ; 或  MOV    AX, [EBX*8+ESI]
```

基址加比例变址寻址适合检索多字节二维数组元素等操作。

（8）带位移量的基址加变址寻址

带位移量的基址加变址寻址方式下，有效地址 EA=(基址寄存器)+(变址寄存器)+位移量。它也分 16 位寻址和 32 位寻址两种情况。基址、变址寄存器的使用规定和对段寄存器的默认规定与前面所述相同。例如：

```
MOV     AX, [BX+SI+MASK]          ; 或  MOV    AX, MASK[BX][SI]
```

带位移量的基址加变址寻址也适合对二维数组操作，位移量即为数组起始地址。

（9）带位移量的基址加比例变址寻址

这种寻址方式用到有效地址的全部 4 个分量，EA=(基址寄存器)+(变址寄存器)×比例因子+位移量。与另外两种含比例因子的寻址方式一样，它只有 32 位寻址一种情况。各种规定和默认情况同前所述。寻址过程中，变址寄存器内容乘以比例因子的操作在 CPU 内部由硬件完成。例如：

```
MOV     AX, [EDI*8+ECX+40]        ; 或  MOV    AX, [EDI*8][ECX+40]
```

带位移量的基址加比例变址寻址适合起始地址不为零的多字节二维数组元素的检索操作。

3.2.3 80x86/Pentium 存储器寻址的段约定

进行存储器操作数访问时，指令中一般不特别指出段寄存器，这是因为对各种类型的存储器寻址，在 80x86/Pentium 中关于段和偏移地址寄存器的基本默认约定如表 3-2 所示。只要在指令中不特别说明要超越这个约定，则按基本约定来寻找操作数。

電子工業出版社

表 3-2　存储器操作数访问中关于段和偏移地址寄存器的约定

访问存储器类型	默认段寄存器	允许超越的段寄存器	偏移地址寄存器
取指令	CS	无	(E)IP
堆栈操作	SS	无	(E)SP
源串数据访问	DS	CS、ES、SS（FS、GS）	(E)SI
目的串数据访问	ES	无	(E)DI
通用数据访问	DS	CS、ES、SS（FS、GS）	偏移地址 EA
以(E)BP、(E)SP 间接寻址的指令	SS	CS、DS、ES（FS、GS）	偏移地址 EA

表 3-2 说明，除了程序只能在代码段、堆栈操作数只能在堆栈段、目的串操作数只能在附加数据段，其他约定都是允许段超越的。

3.2.4　几种处理器寻址方式比较

总结 80x86/Pentium CPU 指令系统的寻址方式，立即寻址与寄存器寻址是最好理解的，存储器寻址分为两大类，即指令中给出的是存储器操作数的绝对地址还是相对地址。

MIPS 或 ARM 等 RISC 处理器的指令条数和寻址方式少，因而指令的译码速度快，控制逻辑规整简单。MIPS 及 ARM 与 80x86/Pentium 指令在寻址方式上的重要不同就是存储器寻址，由于前者存储器操作数的地址分量少，所以它们的寻址方式较为简洁。表 3-3 给出了它们的寻址方式比较。

表 3-3　各种处理器寻址方式比较

		80x86/Pentium	MIPS	ARM
立即寻址		√	√	√
寄存器寻址		√	√	√
存储器寻址方式	直接寻址	√	√	√
	寄存器间接寻址	√		√
	基址寻址	√	√	√
	变址寻址	√		
	比例变址寻址	√		
	基址加变址寻址	√	√	
	基址加比例变址寻址	√		
	带位移量的基址加变址寻址	√		
	带位移量的基址加比例变址寻址	√		
	多寄存器寻址			√
	寄存器移位寻址			√

在 ARM 中，定义了不同工作模式下的多组通用寄存器 R0-R15，它们都可作为基址寄存器。除了与 80x86/Pentium 的一些相同寻址方式，ARM 增加了专有的多寄存器和寄存器移位寻址方式。MIPS 的所有通用寄存器都可以定义为基址寄存器或变址寄存器。例如，ARM 多寄存器寻址：

```
STMIA     R0!,{R1-R4}  ; [R0]←R1, [R0+4]←R2, [R0+8]←R3, [R0+12]←R4
                       ; 将 R1-R4 的内容送到以 R0 为首地址的连续内存区域并更新 R0
```

ARM 寄存器移位寻址：

```
ADD       R0, R1, R2, LSL #1  ; R0←R1+(R2<<1)，将 R2 左移一位后与 R1 相加，结果
```


電子工業出版社

存 R0

MIPS 基址加变址寻址：

```
ADD        $1,($2+$3)    ; R1←R1+[R2+R3]，将内存地址 R2+R3 中的值与寄存器 R1
                         ; 相加，结果保存到 R1
```

3.3 8086/8088 CPU 指令系统

Intel 8086/8088 CPU 指令系统是 80x86/Pentium 系列 CPU 的基本指令集。指令的操作数宽度是 8 位或 16 位，偏移地址宽度是 16 位。可以分成 6 个功能组：数据传送类、算术运算类、逻辑运算与移位类、串操作类、控制转移类和处理器控制类。本节着重介绍这些指令的功能与应用。为方便理解指令，需简要说明指令操作数符号，如表 3-4 所示。

表 3-4　指令操作数符号说明

符　号	意　义
OPRD，OPRD*N*	操作数；第 *N* 个操作数，如 OPRD1、OPRD2、OPRD3 等
DEST，SRC	目的操作数，源操作数
reg	通用寄存器，长度可以是 8 位或 16 位
reg8	8 位通用寄存器 AH/AL/BH/BL/CH/CL/DH/DL
reg16	16 位通用寄存器 AX/BX/CX/DX/SI/DI/BP/SP
Sreg	段寄存器 DS/CS/SS/ES（/FS/GS）
mem	存储器操作数，长度可以是 8 位或 16 位
mem8	8 位存储器操作数
mem16	16 位存储器操作数
imm	立即数，长度可以是 8 位或 16 位
imm8	8 位立即数
imm16	16 位立即数

3.3.1 数据传送类指令

数据传送是计算机中最基本最重要的一类操作，在程序中使用的比例最高。它们用于实现寄存器与寄存器、寄存器与存储器、累加器（AL/AX）与I/O端口之间的字节或字的传送，也可将立即数传送到寄存器或存储器单元中。为此，指令中必须指明操作数的源位置和目标位置。数据传送类指令有14条，分为4组，如表3-5所示。除了SAHF和POPF指令，其余指令执行后对标志位都没有影响。

1．通用数据传送指令

通用数据传送指令共5条，除了XCHG，是唯一允许以段寄存器作为操作数的指令。

（1）MOV 指令

指令格式：MOV　　OPRD1, OPRD2

功能：将源操作数传送给目标操作数，即 OPRD2→OPRD1，源操作数保持不变。OPRD1 和 OPRD2 可以是字节或字，但两者必须等长（同为字节传送或字传送）。

MOV 指令有如下一些形式：

```
MOV   reg/Sreg/mem, reg          ; reg/Sreg/mem←reg
```


電子工業出版社

表 3-5　数据传送类指令

类　别	指令功能	指令书写格式（助记符）
通用数据传送	字节或字传送	MOV　目标，源
	字压入堆栈	PUSH　源
	字弹出堆栈	POP　目标
	字节或字交换	XCHG　目标，源
	字节翻译	XLAT
地址传送	装入有效地址	LEA　目标，源
	装入逻辑地址	LDS　目标，源
	装入逻辑地址	LES　目标，源
标志位传送	将标志寄存器 FR 低字节装入 AH	LAHF
	将 AH 内容装入 FR 低字节	SAHF
	将 FR 内容压入堆栈	PUSHF
	从堆栈弹出 FR 内容	POPF
I/O 数据传送	输入字节或字	IN　累加器，端口
	输出字节或字	OUT　端口，累加器

```
MOV   reg/mem, Sreg          ; reg/mem←Sreg
MOV   reg/Sreg, mem          ; reg/Sreg←mem
MOV   reg/mem, imm           ; reg/mem←imm
```

源操作数可以是通用寄存器、段寄存器、存储器以及立即操作数；目标操作数可以是通用寄存器、段寄存器（CS 除外）或存储器。各种数据传送关系如图 3-4 所示。例如：

```
MOV   AL, CH                 ; 通用寄存器之间传送字节数据
MOV   DS, AX                 ; 通用寄存器与段寄存器（CS 不能是目标）之间传送数据
MOV   AX, 0FF3BH             ; 立即数 FF3BH 传送到通用寄存器
MOV   AL,BUFFER              ; 存储器与通用寄存器之间传送数据，BUFFER 为字节存储单元名
MOV   DAT[BP+DI], ES         ; 段寄存器与存储器之间传送数据，DAT 为常量或字存储单元名
```

使用 MOV 指令传送数据时应注意：① 立即数和段寄存器 CS 不能作为目标操作数；② 立即数不能直接传送到段寄存器；③ 不允许在两个存储单元之间或在两个段寄存器之间直接传送数据；④ 将立即数传送到存储单元时，必须显式说明存储器操作数的宽度类型。

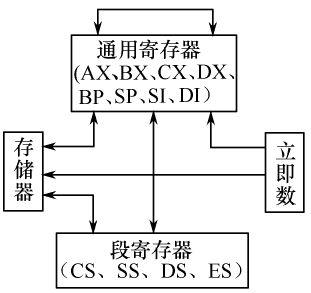


图 3-4　数据传送关系

（2）堆栈操作指令 PUSH/POP

堆栈是内存的一个数据区，按照“后进先出”原则组织的一段内存区域。80x86/Pentium CPU 规定，堆栈设置在堆栈段（SS）内，向下生长，堆栈指针 SP 始终指向堆栈的顶部。在子程序（过程）调用或处理中断时，堆栈用于保存当前的断点地址（在 8086/8088 中为 CS 和 IP）和现场数据，以便程序正确返回。断点地址的保存是由子程序调用指令或中断响应来完成的，现场数据保存可通过堆栈操作指令来实现。

堆栈操作指令有两条：入栈指令 PUSH 和出栈指令 POP。当需要在一个操作数和堆栈之间传送数据时，常使用它们。

指令格式：PUSH　　OPRD2

　　POP　　OPRD1

電子工業出版社

功能：PUSH 指令使 SP−2→SP，然后将 16 位源操作数压入堆栈。入栈时，高字节存放在高地址，低字节存放在低地址。源操作数可以是通用寄存器、段寄存器和存储器。

出栈指令 POP 的执行过程与 PUSH 相反，它从栈顶弹出 16 位操作数到目标操作数。弹出时，低地址字节送低字节，高地址字节送高字节，同时修改 SP+2→SP，使 SP 指向新的栈顶。目标操作数可以是通用寄存器、段寄存器（CS 除外）或存储器。

对 8086/8088 来说，堆栈操作总是按字（而不是字节）存取，具体入栈/出栈指令如下：

```
PUSH  reg16  ; SP=SP-2,[SP]←reg16     POP  reg16   ; reg16←[SP]，SP=SP+2
PUSH  Sreg   ; SP=SP-2,[SP]←Sreg      POP  Sreg    ; Sreg←[SP]，SP=SP+2
PUSH  mem16  ; SP=SP-2,[SP]←mem16     POP  mem16   ; mem16←[SP]，SP=SP+2
```

图 3-5 是指令 PUSH　AX 和 POP　AX 的执行示例。

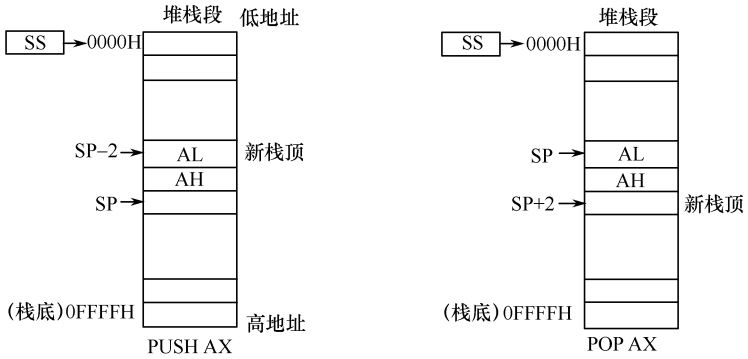


图 3-5　堆栈操作

（3）交换指令 XCHG

指令格式：XCHG　　OPRD1, OPRD2

功能：将长度相同（同为字节或同为字）的源操作数与目标操作数进行交换。

具体指令如下：

```
XCHG  reg/mem, reg        ; reg/mem←→reg
XCHG  reg, mem            ; reg←→mem
```

例如：

```
MOV   AX, 2244H           ; AX=2244H
MOV   BX, 3366H           ; BX=3366H
XCHG  AX, BX              ; 交换后，AX=3366H，BX=2244H
```

注意：① 段寄存器和立即数不能作为操作数；② 不能在两个存储单元之间直接交换数据。

（4）查表转换指令 XLAT（或称为换码指令）

指令格式：XLAT　或　XLAT　　OPRD　　; AL←[BX+AL]

功能：完成 1 字节的查表转换，将数据段中偏移地址为(BX+AL)的存储单元的内容送入 AL 寄存器，即 DS:[BX+AL]→AL。

XLAT 指令对于一些无规律的代码转换（换码）特别方便。使用时，先在数据段建立一个长度小于 256 字节的表格，表的首地址置于 BX 中，AL 中存放查找对象在表中的下标。指令执行后，所查找的对象存于 AL 中，BX 内容保持不变。

例如，将数字 0～9 的 BCD 码转换为 7 段 LED 显示器的显示代码。

数字 0～9 的 BCD 码对应的 7 段 LED 显示代码为 40H，79H，24H，30H，19H，12H，

電子工業出版社

02H，78H，00H，18H，将它们依次存放在数据段中偏移地址为 0800H 开始的区域。若待转换的 BCD 码为 0100B，则实现代码转换的程序段为：

```
MOV   BX, 0800H
MOV   AL, 4
XLAT                        ; AL 中为 0100B 对应的 7 段代码 19H
```

XLAT 指令的操作数是隐含的，汇编语言程序中可指定表的首地址以方便阅读。表的首地址名称仅供汇编程序使用，对指令的执行没有任何影响。该指令还允许段超越，如 ES:XLAT。

2．地址传送指令

地址传送指令用于传送存储器的逻辑地址信息（见表 3-5），指令中的源操作数只能是存储器操作数。

（1）有效地址传送指令 LEA

指令格式：LEA　　OPRD1, OPRD2

功能：将源操作数在当前段内的有效地址（即偏移地址）传送至目标操作数。

具体指令为：

```
LEA     reg16, mem          ; reg16←Addr(mem)
```

注意：这条指令同 MOV 指令的区别是，MOV 指令传送操作数的内容，而 LEA 传送的是操作数的存储地址。例如：

```
MOV     DI, TABLE           ; 将变量 TABLE 的内容传送至 DI
LEA     DI, TABLE           ; 将变量 TABLE 的偏移地址传送至 DI
LEA     DX, [1000H]         ; 将 1000H 单元的偏移地址→DX，DX=1000H
```

（2）地址指针传送指令 LDS 和 LES

指令格式：LDS/LES　　OPRD1, OPRD2

这两条指令的功能类似，都是将由源操作数有效地址决定的双字存储单元中的第一个字的内容送入指令指定的 16 位通用寄存器，第二个字的内容传送给段寄存器 DS 或 ES。通常，双字存储单元中是一个 32 位的地址指针。

具体指令为：

```
LDS（LES）   reg16, mem      ; reg16←[EA]，DS(ES)←[EA+2]
```

例如：

```
TABLE   DD  12345678H
LDS     BX, TABLE           ; BX←5678H，DS←1234H
```

3．标志位传送指令

CPU 通过这类指令对标志寄存器（FR）进行保护与更新。指令的操作数用隐含方式规定，指令共 4 条，见表 3-5。

（1）标志寄存器读/写指令 LAHF/SAHF

指令 LAHF 用于将标志寄存器的低字节（含 SF、ZF、AF、PF、CF）读出后传送给 AH 寄存器。这条指令本身不影响标志位。

标志寄存器写指令 SAHF 与 LAHF 的操作相反。它把寄存器 AH 中的内容写入标志寄存器的低字节，取代某些标志位（SF、ZF、AF、PF、CF）的原来状态。

電子工業出版社

（2）标志寄存器入栈/出栈指令 PUSHF/POPF

PUSHF 指令的执行过程同 PUSH 指令，只是将标志寄存器 FR 的内容压入堆栈。POPF 则将当前栈顶的一个字传送给标志寄存器 FR，同时修改栈顶指针 SP。

在子程序调用和中断服务中，常用这两条指令保护和恢复需要的标志位。利用它们还可以方便地改变标志寄存器中任一位的状态。例如，8086/8088 指令系统中没有直接修改 TF 标志位的指令，可用以下程序段来实现 TF 的改变：

```
PUSHF
POP     AX                  ; 标志寄存器的内容送 AX
OR      AH, 01H             ; 将 TF 位置 1（TF 在标志寄存器的 b8 位）
PUSH    AX
POPF                        ; AX 的内容送标志寄存器
```

4．输入/输出数据传送指令 IN/OUT

指令格式：IN　　OPRD1, OPRD2

　　OUT　　OPRD1, OPRD2

这组指令专门用于在 AL 或 AX 寄存器与 I/O 端口之间传送数据。

一台计算机可以配接许多外部设备，每个外部设备与 CPU 之间要交换数据、状态信息和控制命令，每一种这样的信息交换都要通过一个端口来进行。计算机系统中的各端口也像存储器那样用地址来区分。80x86/Pentium CPU 中有 16 条 I/O 地址线，最多可提供 64K 个传送 8 位数据的端口地址或 32K 个传送 16 位数据的端口地址。当端口地址小于 256 时，采用直接寻址方式，在指令中直接指定端口地址；当端口地址大于或等于 256 时，采用间接寻址方式，端口地址放在 DX 中。

具体指令如下：

```
IN    AL, imm8   ; AL←[端口 imm8](其他同)   OUT  imm8, AL  ; [imm8] ←AL
IN    AX, imm8   ; AX←[imm8+1], [imm8]     OUT  imm8, AX  ; [imm8+1], [imm8] ←AX
IN    AL, DX     ; AL←[DX]                 OUT  DX,AL     ; [DX]←AL
IN    AX, DX     ; AX←[DX+1], [DX]         OUT  DX,AX     ; [DX+1], [DX]←AX
```

例如：

```
IN    AX, 20H             ; 从端口 20H 和 21H 输入 16 位数据到 AL 和 AH(AX)
MOV   DX, 3F0H
IN    AL, DX              ; 从端口 03F0H 输入 8 位数据到 AL
OUT   27H, AL             ; 将 8 位数据从 AL 输出到端口 27H
OUT   DX, AX              ; 将 16 位数据从 AX 输出到 DX 和 DX+1 指定的端口
```

3.3.2　算术运算类指令

算术运算类指令支持加、减、乘、除等基本算术运算，操作对象可以是字节或字的无符号和有符号的二进制整数，也可以是无符号的压缩、非压缩 BCD 数。非压缩 BCD 数的高 4 位在进行乘/除运算时必须全为 0，进行加/减运算时可以是任何值。

算术运算类指令共 20 条，如表 3-6 所示。除了数据宽度变换指令（CBW，CWD），其余指令的执行结果都影响标志位。这些标志位中的绝大多数可供条件转移指令进行测试，以改变程序的流程。掌握指令执行结果对标志位的影响，对编程有重要的作用。

電子工業出版社

1．二进制算术运算指令

（1）加/减法类指令

① 加/减法指令 ADD/SUB

指令格式：ADD　　OPRD1, OPRD2

　　SUB　　OPRD1, OPRD2

表 3-6　算术运算类指令

类　别	指令功能	指令书写格式（助记符）	状态标志位					
			OF	SF	ZF	AF	PF	CF
加法	加法	ADD　目标，源	↕	↕	↕	↕	↕	↕
	带进位加法	ADC　目标，源	↕	↕	↕	↕	↕	↕
	加 1	INC　目标	↕	↕	↕	↕	↕	—
减法	减法	SUB　目标，源	↕	↕	↕	↕	↕	↕
	带借位减法	SBB　目标，源	↕	↕	↕	↕	↕	↕
	减 1	DEC　目标	↕	↕	↕	↕	↕	—
	取负	NEG　目标	↕	↕	↕	↕	↕	①
	比较	CMP　目标，源	↕	↕	↕	↕	↕	↕
乘法	不带符号数乘法	MUL　源	↕	*	*	*	*	↕
	带符号数乘法	IMUL　源	↕	*	*	*	*	↕
除法	不带符号数除法	DIV　源	*	*	*	*	*	*
	带符号数除法	IDIV　源	*	*	*	*	*	*
	字节转换成字	CBW	—	—	—	—	—	—
	字转换成双字	CWD	—	—	—	—	—	—
十进制调整	非压缩 BCD 数的加法调整	AAA	*	*	*	↕	*	①
	压缩 BCD 数的加法调整	DAA	*	↕	↕	↕	↕	↕
	非压缩 BCD 数的减法调整	AAS	*	*	*	↕	*	↕
	压缩 BCD 数的减法调整	DAS	↕	↕	↕	↕	↕	↕
	非压缩 BCD 数的乘法调整	AAM	*	↕	↕	*	↕	*
	非压缩 BCD 数的除法调整	AAD	*	↕	↕	—	↕	*

注：↕ 运算结果影响标志位；* 标志位为任意值；— 运算结果不影响标志位；① 标志位置 1。

功能：完成两个操作数的加/减运算，结果送入目标操作数，即 OPRD1±OPRD2→OPRD1。

具体指令如下：

```
ADD   reg, reg/mem/imm        ; reg←reg + (reg/mem/imm)
ADD   mem, reg/imm            ; mem←mem + (reg/imm)
SUB   reg, reg/mem/imm        ; reg←reg – (reg/mem/imm)
SUB   mem, reg/imm            ; mem←mem – (reg/imm)
```

要求源操作数和目标操作数同为带符号数或同为无符号数，且长度相等。源操作数可以是寄存器、存储器或立即数，目标操作数只能是寄存器或存储器，且两个操作数不能同时为存储器。运算影响全部状态标志位。

例如，设 AX=65A0H，BX=B79EH，则指令 ADD　　BX, AX 执行后，BX=1D3EH，CF=1，SF=0，OF=0，AF=0，PF=0，ZF=0。

结果表明，若 BX 和 AX 中是无符号数，则运算结果大于 65535（CF=1）；若 BX 和 AX 中是有符号数，则运算结果在 16 位补码所能表示的范围内（–32768～+32767），运算结果正确（OF=0，SF=0）。

電子工業出版社

若 AX、BX 中存放的数和上例相同，则指令 SUB　AX, BX 执行后，AX=AE02H，CF=1，SF=1，OF=1，AF=1，PF=0，ZF=0。

结果表明，若两个操作数是无符号数，CF=1 表示不够减，运算结果是以 2^{16} 为模的差值的补码；若两个操作数是有符号数，则运算结果超出 16 位补码所能表示的范围（OF=1），AX 中存放的数（AE02H）并不是正确的运算结果。

② 带进位、借位的加/减法指令 ADC/SBB

指令格式：ADC　OPRD1, OPRD2

SBB　OPRD1, OPRD2

ADC/SBB 指令常与 ADD/SUB 配合，用于多精度数间的运算。除了加法操作时要加上进位位或减法操作时再减去借位位，它们同 ADD/SUB 指令的使用注意事项和对标志位的影响情况相同。

具体指令如下：

```
ADC  reg, reg/mem/imm              ; reg←reg + (reg/mem/imm) + CF
ADC  mem, reg/imm                  ; mem←mem + (reg/imm) + CF
SBB  reg, reg/mem/imm              ; reg←reg – (reg/mem/imm) – CF
SBB  mem, reg/imm                  ; mem←mem – (reg/imm) – CF
```

例如，有两个 32 位数（多精度数）分别存放在自 FIRST 和 SECOND 开始的存储区中，变量定义如下（DW 是汇编语言伪指令，它为变量 FIRST、SECOND 按字分配存储单元）：

```
FIRST    DW  2211H, 4433H          ; 定义变量 FIRST=44332211H
SECOND   DW  6655H, 8877H          ; 定义变量 SECOND=88776655H
```

用字相加指令实现两个变量的和，并将和存放于 FIRST 变量的程序段为：

```
MOV  AX, SECOND            ; 取第二个加数的低 16 位
ADD  FIRST, AX             ; 与第一个加数的低 16 位相加并存回和，如有进位，则 CF=1
MOV  AX, SECOND+2          ; 取 32 位数据 SECOND 的高 16 位
ADC  FIRST+2, AX           ; 将两个数的高 16 位相加并加上 CF，和存回 FIRST 的高 16 位
```

③ 增量和减量指令 INC/DEC

指令格式：INC　OPRD

DEC　OPRD

功能：INC/DEC 是单操作数指令，完成对指定操作数的加 1/减 1，然后返回运算结果。

具体指令如下：

```
INC   reg        ; reg←reg+1          DEC   reg        ; reg←reg-1
INC   mem        ; mem←mem+1          DEC   mem        ; mem←mem-1
```

设计这两条指令的主要目的是调整地址指针和计数器（循环程序的循环次数），所以它们不影响 CF 标志（即保持此指令之前的值）。对其他 5 个标志位的影响与前述加/减法指令一样。指令中的操作数可以在通用寄存器或内存中。例如：

```
INC   CX                           ; CX 寄存器中的内容加 1
DEC   BYTE  PTR [BX]               ; 将数据段中以 BX 指向的字节存储单元内容减 1
DEC   WORD  PTR [BX]               ; 将数据段中以 BX 指向的字存储单元的内容减 1
```

④ 求补指令 NEG

指令格式：NEG　OPRD

功能：对操作数求补，即用零减去操作数，再把结果送回操作数。求补运算也可以表达成：将操作数按位求反加 1。指令执行的效果是改变操作数的符号，但绝对值不变，所以

電子工業出版社

又称为取负指令。具体指令如下：

```
NEG   reg                    ; reg←0-reg
NEG   mem                    ; mem←0-mem
```

NEG 指令影响所有状态标志位，不过对 CF 标志的影响为：若操作数不是 0，则总是使 CF 置 1；否则将 CF 清 0。若在字节操作时对−128、或在字操作时对−32768 求补，则操作数没有变化，但溢出标志 OF 置位。例如：

```
NEG   AL                     ; 这两条指令实现(100-AL)运算
ADD   AL, 100
```

⑤ 比较指令 CMP

指令格式：CMP　　OPRD1, OPRD2

比较指令 CMP 计算 OPRD1−OPRD2，但运算结果不送回 OPRD1，即 CMP 指令与减法指令 SUB 执行同样的操作，同样影响标志位，只是不改变操作数本身。参加比较的源操作数可以是立即数、寄存器或存储器操作数，目标操作数只能是寄存器或存储器操作数。

具体指令为：

```
CMP   reg, reg/mem/imm       ; reg-(reg/mem/imm)
CMP   mem, reg/imm           ; mem-(reg/imm)
```

比较指令用于比较两个数的大小关系，执行比较指令后，可以根据受影响的标志位状态来判断两个操作数是否相等、谁大、谁小。所以 CMP 指令后面常跟条件转移指令，根据比较结果产生不同的分支转移。判断无符号数和有符号数大小的条件有所不同，列举如下。

- 如果是两个无符号数比较，可根据比较结果的进位标志 CF 的状态来判断：若 CF=1，则 OPRD1<OPRD2；若 CF=0，则 OPRD1>OPRD2。
- 如果是两个有符号数比较，则根据 SF 和 OF 两个标志位的关系来判断：若 SF⊕OF=1，则 OPRD1<OPRD2；若 SF⊕OF=0，则 OPRD1>OPRD2。
- 不管操作数有无符号，若在比较指令后，ZF=1，则 OPRD1=OPRD2，否则不等。

所以，条件转移指令也按无符号数和有符号数分为两类（详见表 3-11）。

（2）乘/除法指令

① 乘法指令 MUL/IMUL

指令格式：MUL　　OPRD

　　IMUL　　OPRD

MUL 和 IMUL 指令分别用于实现无符号数的乘法和有符号数的乘法运算。它们都只有一个源操作数，源操作数可以是寄存器或存储器，而目标操作数隐含规定在累加器中。

具体的指令如下：

```
MUL   reg    或   IMUL reg     ; AX←AL × reg8（或 DX,AX←AX × reg16）
MUL   mem    或   IMUL mem     ; AX←AL × mem8（或 DX,AX←AX × mem16）
```

MUL 指令用于无符号二进制数乘法运算。若为字节（OPRD 为字节）乘法，则 AL×OPRD，乘积存于 AX 中；若为字（OPRD 为字）乘法，则 AX×OPRD，乘积的高 16 位存放在 DX 中、低 16 位存放在 AX 中。当乘积的高半部分（字节相乘时为 AH，字相乘时为 DX）不为 0，则标志 CF=OF=1，表示在 AH 或 DX 中存有乘积的有效数值；否则，CF=OF=0。而其他状态标志位的内容不确定。

IMUL 指令除了完成两个有符号数的相乘，其他与 MUL 完全类似。若乘积的高半部分（字节相乘时为 AH，字相乘时为 DX）存有积的有效数值（不光是符号部分），则 CF=OF=1；

電子工業出版社

否则 CF=OF=0，表示高半部分全部是低半部分符号位的扩展。

由于乘法指令为乘积保留了 2 倍于原来操作数的存储空间，因而不会出现溢出现象。

② 除法指令 DIV/IDIV

指令格式：DIV　　OPRD

　　IDIV　　OPRD

DIV 和 IDIV 分别用来实现无符号数的除法和有符号数的除法运算。指令中对操作数的规定同乘法指令，除法指令执行后，所有的状态标志位都不确定。

具体的指令如下：

```
DIV   reg    或   IDIV   reg        ; AH,AL←AX/reg8（或 DX,AX←DX:AX/reg16）
DIV   mem    或   IDIV   mem        ; AH,AL←AX/mem8（或 DX,AX←DX:AX/mem16）
```

DIV 用于无符号数的除法运算。对字节除法，被除数在 AX 中，除数即 OPRD（字节），商在 AL 中，余数在 AH 中；对字除法，被除数高位在 DX 中、低位在 AX 中，除数即 OPRD（字），商在 AX 中，余数在 DX 中。若商超过存放它的寄存器的容量（字节相除时为 FFH，字相除时为 FFFFH），则引起 0 型中断（即除法出错中断），且商和余数为不定值。

IDIV 除了完成有符号数相除，其余与 DIV 完全类似。商的符号根据代数符号的规则确定，余数的符号与被除数的相同。在字节相除时，最大的商是+127（7FH），最小的负数商是−127（81H）；在字相除时，最大的商为 32767（7FFFH），最小的负数商是−32767（8001H）。若相除以后，商超出上述讨论值的范围，则商和余数的值不能确定，并产生一个 0 型中断。

注意：在除法运算中，要求被除数的长度为除数长度的 2 倍。若被除数和除数是具有相同位数的字节或字，这时需要先将被除数扩展为字或双字，且大小和符号不变，再相除。

2．符号扩展指令 CBW/CWD

指令格式：CBW

　　CWD

功能：CBW 和 CWD 分别用于将有符号操作数从字节转换成字或从字转换为双字，操作数隐含规定在累加器中。CBW 将 AL 中的字节符号数的符号扩展到 AH 中，CWD 将 AX 中的字符号数的符号扩展到 DX 中。CBW 和 CWD 指令都不影响任何标志位。

扩展规则如下：① 若 AL<80H，则 AH←0；若 AL≥80H，则 AH←FFH；② 若 AX<8000H，则 DX←0；若 AX≥8000H，则 DX←FFFFH。

例如，求 0BF4H÷0100H（有符号数相除）。由于除数为字，则必须将被除数进行符号扩展后才能相除。程序段如下：

```
MOV    AX, 0BF4H
CWD                                ; 被除数扩展为 DX:AX=0000 0BF4H
MOV    BX, 0100H
IDIV   BX                          ; AX←商 0BH，DX←余数 F4H
```

注意：对无符号数，可采用高位直接置“0”的方法进行扩展。

3．BCD（十进制）运算调整指令

CPU 中的 ALU 单元只能执行二进制数的算术操作。对于 BCD 数的算术运算，可以直接使用二进制数的算术运算指令，但必须在二进制算术运算指令后，紧接着用一条调整指令来加以校正调整。所有调整指令都隐含操作数在累加器中，对标志位的影响如表 3-6 所示。

（1）非压缩 BCD 数加/减法调整指令 AAA/AAS

AAA/AAS 用于将 AL 中的由两个非压缩 BCD 数相加/减后的结果进行调整，以得到正

電子工業出版社

确的非压缩 BCD 和/差。程序中，它们紧跟在 ADD（或 ADC）/SUB（或 SBB）指令后。

AAA/AAS 指令完成如下调整操作：若(AL&0FH)>9 或标志 AF=1，则 AL←AL±6，AH←AH±1，AF←1，CF←AF，AL←AL&0FH（“-”对指令 AAS 而言）。

（2）压缩 BCD 数加/减法调整指令 DAA/DAS

DAA/DAS 用于将 AL 中两个压缩 BCD 数相加/减的结果进行校正，以得到正确的压缩 BCD 和/差。程序中，它们紧跟在 ADD（或 ADC）/SUB（或 SBB）指令后。

DAA/DAS 指令完成如下调整操作：若(AL&0FH)>9 或 AF=1，则 AL←AL±6，AF←1（“-”对 DAS 而言）；若 AL>9FH 或 CF=1，则 AL←AL±60H，CF←1（“-”对 DAS 而言）。

例如，AL=47H（十进制数 47 的 BCD 码），BH=25H（十进制数 25 的 BCD 码），用下列指令可得到这两个数和的正确结果。

```
ADD   AL, BH              ; AL←AL+BH
DAA                       ; 将 AL 的内容调整为 72H
```

注意，所有调整指令仅用来对一次加/减运算的结果 AL 进行调整。因而对多位压缩或非压缩的 BCD 数的运算，必须由低位向高位逐个进行调整。

【例 3-1】 计算$(8576)_{10}+(2695)_{10}$。假设两个操作数分别存于 AX、BX，即 AX=8576H，BX=2695H，则下列程序段将实现两个数的求和运算，结果放在 AX 中。

```
ADD   AL, BL              ; 低位相加
DAA                       ; 结果调整并存于 AL 中
MOV   DL, AL              ; 暂存低字节压缩 BCD 数在 DL 中
MOV   AL, AH
ADC   AL, BH              ; 高位相加
DAA                       ; 结果调整并存于 AL 中
MOV   AH, AL              ; 高字节压缩 BCD 数在 AH 中
MOV   AL, DL              ; 压缩 BCD 数的和在 AX 中
```

（3）非压缩 BCD 数乘/除法调整指令 AAM/AAD

AAM 紧跟在 MUL 指令后，将 AX 中两个 1 位非压缩 BCD 数相乘的结果进行调整，得到正确的非压缩 BCD 的乘积（高位在 AH 中，低位在 AL 中）。调整操作为：

```
AH ← AX/0AH               ; AX 被 0AH 除的商→AH
AL ← AX % 0AH             ; AX 被 0AH 除的余数→AL
```

例如：

```
MOV   AL, 07H
MOV   BL, 09H
MUL   BL                  ; 执行乘法指令后 AX=003FH
AAM                       ; 执行乘法调整指令后 AX=0603H
```

AAD 指令用来在进行两个非压缩 BCD 数的除法运算之前，先调整 AL 和 AH 中的内容，再用二进制除法指令 DIV 相除。相除以后，以非压缩 BCD 数表示的商在 AL 中，相应的余数在 AH 中。调整操作为：

```
AL←AH×0AH+AL, AH←0
```

例如：设 AX=0103H，BL=06H（分别表示非压缩 BCD 数 13 和 6）。

```
AAD                       ; 先调整得 AX=000DH
DIV   BL                  ; 执行除法指令后，商 AL=02H，余数 AH=1
```


電子工業出版社

3.3.3 逻辑运算与移位指令

为了处理字节或字中各位的信息，8086/8088 CPU 提供了一组位处理指令，包括逻辑运算、移位和循环移位指令三部分。

1. 逻辑运算指令

逻辑运算指令列于表 3-7 中。所有的指令都对操作数进行按位操作，操作数可以是字节或字。目标操作数不能是立即数；当有两个操作数时，不能两个操作数都为存储器操作数；无论是目标操作数还是源操作数都不能是段寄存器。

表 3-7 逻辑运算类指令

类 别	指令功能	指令书写格式（助记符）	状态标志位					
			OF	SF	ZF	AF	PF	CF
逻辑运算	非（字/字节）	NOT 目标	—	—	—	—	—	—
	与（字/字节）	AND 目标，源	0	↕	↕	*	↕	0
	或（字/字节）	OR 目标，源	0	↕	↕	*	↕	0
	异或（字/字节）	XOR 目标，源	0	↕	↕	*	↕	0
	测试（字/字节）	TEST 目标，源	0	↕	↕	*	↕	0

注：↕ 运算结果影响标志位；— 运算结果不影响标志位；0 标志位置‘0’；* 标志位为任意值。

逻辑操作指令主要用于，根据源操作数中的位组合格式有选择地使目标操作数的某些位置位、复位，或对其进行测试等。

在这组指令中，仅 NOT 指令不影响标志位。其他指令执行后，除 AF 状态不确定外，总是使 OF=CF=0。ZF、PF、SF 根据运算结果被置位或复位，以反映操作结果的特征。

（1）逻辑非指令 NOT

指令格式：NOT OPRD

功能：将操作数“按位取反”，然后送回原处。

具体指令为：

```
NOT   reg/mem
```

例如：

```
NOT   AX                    ; 对 AX 内容按位取反并送回 AX
```

（2）逻辑与/或/异或指令 AND/OR/XOR

指令格式：AND/OR/XOR OPRD1, OPRD2

功能：AND、OR、XOR 指令执行按位逻辑“与”“或”和“异或”操作。它们均为双操作数指令，两个操作数宽度必须相等（同为字节或字）。执行结果存入 OPRD1 中。

具体指令为：

```
AND/OR/XOR    reg, reg/mem/imm
AND/OR/XOR    mem, reg/imm
```

AND 指令用于对某些位清 0，其余位保持不变（要清 0 的位和 0“与”）；OR 指令用于对某些位置 1，其余位保持不变（要置 1 的位和 1“或”）；XOR 指令使某些位取反，其余位保持不变（要取反的位与 1“异或”）。

特别地，当某个操作数自己和自己相“与”、自己和自己相“或”时，可使 CF 清 0 而

電子工業出版社

操作数保持不变；当操作数自己与自己相“异或”时，可以将操作数和 CF 同时清 0。例如：

```
        XOR   AL, AL          ; 清 AL 寄存器和 CF 位
        MOV   AL, '8'         ; 8 的 ASCII 值送 AL，AL=38H
        AND   AL, 0FH         ; 执行后，AL=08H（AL 的高 4 位被屏蔽，低 4 位被析取）
```

（3）测试指令 TEST

指令格式：TEST　OPRD1, OPRD2

若希望在不改变原有操作数的情况下检测操作数的某一位或某几位是 1 还是 0，可使用 TEST 指令。TEST 指令的功能与 AND 指令相同，但操作结果不送回目标操作数，只根据结果设置标志位。这条指令之后一般都是条件转移指令，利用测试结果转向不同的程序段。

具体指令为：

```
        TEST  reg, reg/mem/imm
        TEST  mem, reg/imm
```

例如，检测 AL 中最低位是否为 1，为 1 则转移至 THERE，可使用以下指令。

```
        TEST  AL, 01H         ; 将源操作数设置为一个立即数，其中需要测试的位置 1，其余位为 0
        JNZ   THERE           ; ZF=0，则 AL 寄存器的第 0 位为 1，转移到 THERE
        …                     ; ZF=1，表明 AL 寄存器的第 0 位为 0，顺序执行
THERE:  …
```

2．移位和循环移位指令

移位和循环移位指令可以对字节或字中的各位进行算术移位、逻辑移位或循环移位，指令及其对应的操作功能如表 3-8 所示。

表 3-8　移位与循环移位指令

类别	指令功能	指令书写格式（助记符）	操作功能	状态标志位					
				OF	SF	ZF	AF	PF	CF
移位	逻辑左移	SHL　目标，计数值	C ← M ← 0	↕	↕	↕	*	↕	↕
	算术左移	SAL　目标，计数值	C ← M ← 0	↕	↕	↕	*	↕	↕
	逻辑右移	SHR　目标，计数值	C → 0 M →	↕	↕	↕	*	↕	↕
	算术右移	SAR　目标，计数值	C → M →	↕	↕	↕	*	↕	↕
循环移位	循环左移	ROL　目标，计数值	C ← M ←	↕	—	—	*	—	↕
	循环右移	ROR　目标，计数值	C → M →	↕	—	—	*	—	↕
	带进位循环左移	RCL　目标，计数值	C ← M ←	↕	—	—	*	—	↕
	带进位循环右移	RCR　目标，计数值	C → M →	↕	—	—	*	—	↕

指令中的目标操作数只能是寄存器或存储器操作数，可以是字节也可以是字。指令中的计数值决定移位或循环移位的次数，计数值或者是 1，或者是 CL 中设定的次数。

（1）移位指令 SHL/SAL/SHR/SAR

移位指令包括逻辑移位与算术移位。从表 3-8 可以看出，逻辑左移与算术左移操作在物理上完全相同，每移位 1 次目标操作数按位依次左移 1 位，最高位进入 CF，右端补 0。逻

電子工業出版社

辑右移 1 次使操作数的左端移入 0，算术右移 1 次则将最高位复制 1 位以保持符号位不变，移出的位都进入 CF。

移位指令影响 PF、SF、ZF、CF、OF 这 5 个标志位。其中，CF 总是等于目标操作数最后移出的那一位。若移位计数值=1，且执行结果使目标操作数的符号位发生变化，则 OF=1，否则 OF=0；若移位计数值>1，则 OF 状态不确定。移位操作对 AF 状态的影响不确定。

具体指令为：

```
SHL/SAL/SHR/SAR    reg，1/CL
SHL/SAL/SHR/SAR    mem，1/CL
```

移位指令用于将字节或字的某些位分离出来，或用来对二进制数进行 2 的方幂运算。左移 1 次，只要左移之后的数值未超出 1 字节或字的表达范围，则原数的每 1 位的权增加一倍，相当于原数乘以 2；右移 1 位相当于除以 2。若需要将有符号数乘以或除以 2^n，则使用算术移位指令；若需要将无符号数乘以或除以 2^n 时，则使用逻辑移位指令。

【例 3-2】 将 AL 中的一个 8 位有符号数乘以 8（2^3），结果存入 AX 中，指令序列为：

```
CBW                                ; 将字节 AL 扩展到字 AX
MOV   CL，3                        ; 移动 3 次，需要将 3 放入 CL
SAL   AX，CL（或 SHL   AX，CL）    ; AX←AX×8
```

【例 3-3】 下列指令将存于 AH 和 AL 中的非压缩 BCD 数转换成压缩的 BCD 数：

```
MOV   CL，4
SHL   AL，CL
SHR   AX，CL                       ; 转换结果在 AL 寄存器中
```

（2）循环移位指令 ROL/ROR/RCL/RCR

循环移位指令有带进位位、不带进位位两种。带进位位的循环移位指令把 CF 标志作为目标操作数的扩展，参与循环操作。与移位指令不同的是，从操作数一端移出来的位“循环地”进入该操作数的另一端。循环移位指令只影响 CF 和 OF 两个标志位，CF 只存放最后一次循环移出的那一位的值，OF 状态的变化规则同移位指令。

具体指令为：

```
ROL/ROR/RCL/RCR    reg，1/CL
ROL/ROR/RCL/RCR    mem，1/CL
```

【例 3-4】 有一个 4 字节（32 位）的数，它们或是存放在两个寄存器中（如 DX 和 AX 中），或是存放在连续的内存单元中，用下列指令可实现这个 4 字节数整体左移 1 位。

```
SAL   AX，1        或        SAL   FIRST_WORD，1
RCL   DX，1        或        RCL   SECOND_WORD，1
```

【例 3-5】 内存中 4 位十进制数以压缩 BCD 码形式存放在 DA1 开始的两个单元，以下程序段实现它们乘以 10 的运算，结果存放在 DA2 开始的内存单元。

```
        MOV   AX，DA1                  ; 取 4 位十进制数→AX
        XOR   DL，DL                   ; 乘积的最高位 DL 清 0
        MOV   CX，4                  ; 4 位十进制数乘 10 就是左移 4 次，CX 中为移位次数
LP:     SHL   AX，1                  ; DL、AX 整体左移 4 次
        RCL   DL，1
        DEC   CX                     ; 移位次数-1
        JNZ   LP                     ; CX≠0，转移到 LP 处继续移位；CX=0，移位结束
        MOV   DA2，AX                ; 保存结果
```


電子工業出版社

```
        MOV   DA2+2, DL
```

3.3.4 串操作指令

数据串是存储器中连续存放的一串字节或字的序列。串操作是对串中的每一项都执行的操作，如串的传送、查找、比较等。8086/8088 指令系统中的专用数据串指令为这些操作提供了很大的方便，如表 3-9 所示。串操作指令可以对字节串或字串进行操作，每次处理串中的一个元素（1 字节或 1 个字），可以处理的数据串长度最大为 64 KB。

表 3-9 串操作指令

类 别	指令功能	指令书写格式（助记符）	状态标志位					
			OF	SF	ZF	AF	PF	CF
基本串操作指令	字节串/字串传送	MOVS 目标串, 源串 MOVSB/MOVSW	—	—	—	—	—	—
	字节串/字串比较	CMPS 源串, 目标串 CMPSB/CMPSW	↕	↕	↕	↕	↕	↕
	字节串/字串搜索	SCAS 目标串 SCASB/SCASW	↕	↕	↕	↕	↕	↕
	读字节串/字串	LODS 源串 LODSB/LODSW	—	—	—	—	—	—
	写字节串/字串	STOS 目标串 STOSB/STOSW	—	—	—	—	—	—
重复前缀	无条件重复	REP	—	—	—	—	—	—
	当相等/为零时重复	REPE/REPZ	—	—	—	—	—	—
	当不等/不为零时重复	REPNE/REPNZ	—	—	—	—	—	—

基本串操作指令有 5 条：传送（MOVS）、比较（CMPS）、搜索（SCAS）、取（LODS）和存（STOS）。任何一个这样的基本操作，能在指令的前面加一个重复前缀使它们的操作重复执行，这就使得处理长数据串比用软件循环进行处理快得多。

所有的串操作指令都有以下共同点：

① 约定以 DS:SI 寻址源串，以 ES:DI 寻址目标串，所以指令中不必显式地指明操作数。其中，源串的段寄存器 DS 可通过加段超越前缀而改变，但目标串的段寄存器 ES 不能超越。

② 用方向标志 DF 规定串处理方向。若 DF=0（用指令 CLD 设置），从低地址向高地址方向处理串的每个元素；若 DF=1（用指令 STD 设置），则处理方向相反。

③ 源、目标串的两个地址指针 SI 和 DI 在每次操作后都将根据方向标志 DF 的值自动增量（DF=0 时）或减量（DF=1 时），以指向串的下一个元素，增量/减量的大小由串元素的长度来决定：字节串时 SI 和 DI 加/减 1，字串时 SI 和 DI 加/减 2。

④ 若在基本串操作指令前加上重复前缀，则可以实现串操作的重复执行。这时必须用 CX 作为重复次数计数器，存放数据串的长度（串元素个数），串操作指令以 CX 为递减计数器自动循环执行 CX 次。

⑤ 重复的数据串处理过程可被中断。CPU 是在处理数据串的下一元素之前识别中断并转入中断服务程序的，因而在中断返回以后，重复过程从中断点继续执行下去。

除了串比较指令（CMPS）和串搜索指令（SCAS），其余串操作指令均不影响标志位。

電子工業出版社

1．重复前缀 REP/REPE/REPNE/REPZ/REPNZ

重复前缀不能单独使用，只能加在串操作指令之前，使基本串操作指令得以重复执行。

（1）在 MOVS、LODS、STOS 指令前加上前缀 REP 后（因为 MOVS、LODS、STOS 不影响标志位，所以这些指令前只能加 REP 指令），则按下列步骤不断地重复执行。

① 若 CX=0，则停止重复过程（退出 REP 操作），否则继续。

② CX←CX−1；执行前缀（REP）后面的串操作指令。

③ 重复①～②。

（2）将 REPE/REPZ 加在 CMPS 或 SCAS 指令之前时，上述重复执行步骤①变为：

① 若 CX=0 或 ZF=0，则停止重复过程，否则继续。

（3）将 REPNE/REPNZ 加在 CMPS 或 SCAS 指令之前时，上述重复执行步骤①变为：

① 若 CX=0 或 ZF=1，则停止重复过程，否则继续。

2．基本串操作指令

（1）串传送指令 MOVS/MOVSB/MOVSW

指令格式：MOVS　　OPRD1, OPRD2

　　MOVSB　　；字节串传送

　　MOVSW　　；字串传送

功能：MOVS 指令把 DS:SI 指定的源串中的 1 字节或 1 字，传送至由 ES:DI 指定的目标串，且根据方向标志 DF 自动地修改指针 SI、DI，以指向串中下一个元素。OPRD1、OPRD2 分别为目标串和源串的符号地址，它们的类型要求一致，并以其类型确定是字节（MOVSB）还是字（MOVSW）操作。

MOVSB、MOVSW 分别为字节串或字串传送指令，不需带操作数。

其他串操作指令都具有这三种格式，其中的符号地址用于确定串的类型并增加可读性。

MOVS 指令的具体形式为：

```
        MOVS    mem, mem        ; [ES:DI]←[DS:SI]，SI←SI±1/±2，DI←DI±1/±2
```

显然，MOVS 指令与 MOV 指令不同，可以实现内存单元之间的数据传送。

例如，将数据段 DATA 内起始地址在 SOURCE（段内偏移地址）的一数据块传送到同一段内起始地址在 DEST 的存储单元中，可用下面的程序段来实现（设数据块长度=100）。

```
        MOV     AX, DATA        ; 设置段寄存器
        MOV     DS, AX
        MOV     ES, AX          ; ES 装入和 DS 相同的值，使附加段和数据段完全重叠
        LEA     SI, SOURCE      ; SOURCE 在数据段
        LEA     DI, DEST        ; DEST 在附加段
        MOV     CX, 100         ; 设置循环操作次数
        CLD                     ; 设置方向标志 DF=0
AGAIN:  MOVS    DEST, SOURCE
        DEC     CX              ; 循环次数-1
        JNZ     AGAIN               ; CX≠0 继续循环操作；CX=0 循环结束
```

若采用重复前缀，则上述程序简化为：

```
        …                       ; 此处 5 条语句同上
        MOV     CX, 100         ; 设置循环操作次数
```


電子工業出版社

```
        CLD                             ; 设置方向标志 DF=0
REP     MOVS    DEST, SOURCE            ; 自动重复至 CX=0
```

（2）串比较指令 CMPS/CMPSB/CMPSW

指令格式：CMPS　　OPRD1, OPRD2

　　CMPSB　　；字节串比较

　　CMPSW　　；字串比较

功能：CMPS 将 DS:SI 指定的源串中的元素减去由 ES:DI 指定的目标串中的相应元素，但两个存储单元中的内容不变，而是用标志位的变化表示比较结果。同时根据方向标志 DF 自动修改源和目标串指针 SI、DI。OPRD1、OPRD2 分别为源串和目标串的符号地址，其类型确定是字节（CMPSB）还是字（CMPSW）操作。

具体指令为：

```
        CMPS    mem, mem            ; [DS:SI]- [ES:DI]，SI←SI±1/±2，DI←DI±1/±2
```

若 CMPS 指令前加上前缀 REPE/REPZ，则表示：当串未结尾（CX≠0）且串相等（ZF=1）时继续比较。若 CMPS 加上 REPNE/REPNZ 前缀，则表示：当串未结尾（CX≠0）且串不相等（ZF=0）时继续比较。

CMPSB/CMPSW 分别表示字节串或字串比较指令，不需带操作数。

例如，利用 CMPS 指令对 STRING1 和 STRING2 两个字符串进行比较。

```
        LEA     SI, STRING1         ; 设 STRING1 在 DS 段
        LEA     DI, STRING2         ; 设 STRING2 在 ES 段
        MOV     CX, COUNT
        CLD                         ; 地址增量比较
REPE    CMPSB                       ; 重复比较，直到不相等（ZF=0）或比较完（CX=0）才退出
        JNE     UNMAT               ; 串不相等，在 RESULT 单元中置 0FFH
        MOV     AL, 0               ; 串相等，在 RESULT 单元中置 0
        JMP     OUTPT               ; 转移到 OUTPT 处
UNMAT:  MOV     AL, 0FFH
OUTPT:  MOV     RESULT, AL          ; 比较结果存于 RESULT
        HLT
```

（3）串搜索指令 SCAS/SCASB/SCASW

指令格式：SCAS　　OPRD

　　SCASB　　；搜索字节串

　　SCASW　　；搜索字串

功能：SCAS 指令将累加器（AL 或 AX）中的内容（关键字）与 ES:DI 指定的目标串元素（字节或字）进行比较（减法操作），用标志位反映比较的结果，而不改变累加器和目标串的内容，达到字符串搜索的目的。同时自动修改指针 DI。OPRD 是目标串的符号地址。

具体指令为：

```
        SCAS    mem                 ; AL/AX - [ES:DI]，DI←DI±1/±2
```

在 SCAS 指令前加 REPE/REPZ 前缀，则表示：当串未结尾（CX≠0）且串元素=关键字（ZF=1）时继续搜索。这种格式用来搜索与给定关键字不同的内容。

在 SCAS 指令前加 REPNE/REPNZ 前缀，则表示：当串未结尾（CX≠0）且串元素≠关键字（ZF=0）时继续搜索，直到 CX=0 或 ZF=1 才执行下一条指令。这种格式可以在一个

電子工業出版社

串中查出某一指定元素。

同样，SCASB/SCASW 分别表示字节串或字串搜索指令。

【例 3-6】搜索附加段的某一数据块 BLOCK 中是否有关键字 KEY？若有，将搜索次数记下来（若次数为 0，表示未搜索到关键字），且记录下存放关键字的地址。程序段如下：

```
        LEA     DI, BLOCK          ; 设置数据块的地址指针
        MOV     CX, COUNT          ; 数据块长度设在 CX 中
        MOV     AL, KEY            ; 关键字送入 AL
        CLD
REPNE   SCASB                      ; ZF=0（不相等）且 CX≠0 重复搜索
        JZ      FOUND              ; ZF=1（相等）搜索到 KEY，转去 FOUND
        MOV     DI, 0              ; 串结束且不相等（ZF=0 且 CX=0），即找不到 KEY，0→DI
        JMP     DONE
FOUND:  DEC     DI
        MOV     POINTR, DI         ; 关键字地址→POINTR 单元
        LEA     BX, BLOCK
        SUB     DI, BX
        INC     DI                 ; 找到 KEY，DI 中为搜索次数
DONE:   HLT
```

（4）串元素存取指令 STOS/STOSB/STOSW 及 LODS/LODSB/LODSW

指令格式：STOS/LODS　　OPRD

　　STOSB/LODSB　　；存/取字节串

　　STOSW/LODSW　　；存/取字串

STOS 将累加器（AL 或 AX）的内容存入由 ES:DI 指定的目标串中，同时自动修改 DI。OPRD 是目标串的符号地址。指令前可加重复前缀 REP，以给内存块赋同一个值。

LODS 将 DS:SI 指定的源串中的元素，传送到 AL（字节操作）或 AX（字操作）寄存器中，同时自动修改 SI。OPRD 是源串的符号地址。这个指令前一般不加重复前缀，因为每重复一次，累加器的内容就要改写一次。

具体指令为：

```
        STOS    mem                ; [ES:DI]←AL/AX，DI←DI±1/±2
        LODS    mem                ; AL/AX←[DS:SI]，SI←SI±1/±2
```

STOSB/STOSW、LODSB/LODSW 的操作分别指定字节和字的 STOS 和 LODS。

3.3.5 控制转移类指令

80x86/Pentium CPU 中，程序的执行序列是由 CS 和 IP 来确定的。CS 包含当前指令所在代码段的段地址信息，IP 中将要执行的下一条指令的偏移地址。程序的执行一般依据指令序列顺序执行，但有时需要改变程序的流程，使程序发生转移（如实现分支、循环、子程序调用等结构）。控制转移类指令的功能就是通过修改 CS 和 IP 寄存器的值，来改变程序的执行顺序，包括：无条件转移指令、调用/返回指令、条件转移指令、循环控制指令和中断指令，如表 3-10 所示。除了中断指令，其他转移指令都不影响状态标志位。

電子工業出版社

表 3-10　控制转移类指令

类　别	指令功能	指令书写格式（助记符）	
无条件转移	无条件转移	JMP	目标标号
过程调用/返回	过程调用	CALL	过程名
	过程返回	RET	[弹出值]
条件转移	根据转移条件 cc 转移到目标标号（共 18 条，详见表 3-11）	Jcc	目标标号
循环控制	循环	LOOP	目标标号
	等于/结果为 0，则循环	LOOPE/LOOPZ	目标标号
	不等于/结果不为 0，则循环	LOOPNE/LOOPNZ	目标标号
	CX 内容为 0，则转移	JCXZ	目标标号
中　断	中断	INT	中断类型
	溢出时中断	INTO	
	中断返回	IRET	

控制转移类指令使程序转移到新的目标，从目标地址处开始执行那里的指令。如果转移的目标地址在当前代码段内，则目标属性为 NEAR，称为段内（近）转移，这时指令只修改 IP；如果目标地址在其他代码段内，则目标属性为 FAR，称为段间（远）转移，这时指令同时修改 IP 与 CS。

无论是段内转移还是段间转移，都有直接转移和间接转移之分。直接转移是转移的目标地址直接出现在指令中；间接转移则是转移的目标地址存储在指令的寄存器或内存变量中。

1．无条件转移指令 JMP

指令格式：JMP　　OPRD

JMP 指令无条件地控制程序转移到 OPRD 所指定的目标地址。它有以下 4 种形式。

（1）段内直接转移

```
    JMP   lable                    ; IP←IP+位移量
```

功能：无条件地转移到 lable 所指定的目标地址。

段内直接转移属于相对转移。指令中的位移量是指目标地址与 JMP 指令的下一条指令地址之差，即跳转地址的偏移范围。如果位移量可以用 1 字节（−128～+127）表达，称为短（SHORT）转移；如果位移量用一个 16 位有符号数（−32768～+32767）表达，则形成近（NEAR）转移。例如：

```
    JMP   short_lable  或   JMP  SHORT 2000H    ; 段内直接短转移，2 字节指令
    JMP   near_lable   或   JMP  2000H          ; 段内直接近转移，3 字节指令
```

通常，汇编程序能够根据位移量的大小自动形成短转移或近转移指令。同时，汇编程序也提供短转移 SHORT 和近转移 NEAR PTR 操作符。

（2）段内间接转移

```
    JMP   reg/mem                  ; IP ← reg / [EA]
```

功能：无条件转移到由寄存器内容指定的目标地址，或是由存储器寻址方式提供的存储单元内容所指定的目标地址。这是一种绝对转移指令，IP 内容被绝对地址替换。例如：

```
    MOV   BX, 1000H
    JMP   BX                       ; 程序将转向 1000H，即 IP←1000H
或  JMP   WORD  PTR [BX+20H]  ; 用 WORD PTR 限定存储单元的类型
```

设 DS=2000H，[21020H]=34H，[21021H]=12H，则第二个 JMP 指令将程序转向 1234H，

電子工業出版社

即 IP=1234H。

（3）段间直接转移

```
JMP   far_lable          ; IP←far_lable 的偏移地址，CS←far_lable 的段地址
```

功能：无条件转移到指定段的目标地址 far_lable。属于绝对转移指令，CS 与 IP 的内容被指令中目标标号 far_lable 的段地址与偏移地址直接替换。例如：

```
JMP    Lable_Decleared_Far  或  JMP    2000:3000H    ; 段间直接转移，5 字节指令
JMP    FAR PTR lable        ; 用 FAR PTR 操作符限定标号 lable 的属性为 FAR
```

或者

```
CODE1  SEGMENT                    |        CODE2    SEGMENT
         …                        |                 …
       JMP     LB1                |        LB1:     …
         …                        |                 …
CODE1  ENDS                       |        CODE2    ENDS
```

（4）段间间接转移

```
JMP   mem                ; IP←[EA]，CS←[EA+2]
```

功能：程序将转向由 mem 操作数提供的双字存储单元内容所指定的目标地址，其中低位字单元内送 IP，高位字单元内容送 CS。例如：

```
MOV    SI, 0100H          ; 新的目标地址在 DS:0100H 开始的连续两个字单元中;
JMP    DWORD  PTR [SI]    ; 其中，IP=WORD PRR[DS:0100H]，CS=WORD PTR[DS:0102H]
```

2．调用与返回指令

在 80x86/Pentium 指令集中，调用子程序（或过程）和从子程序（或过程）返回的指令是 CALL 和 RET。CALL 指令用在调用程序中，RET 用在被调用程序（子程序或过程）中。

执行 CALL 指令时，CPU 首先将其下一条指令（断点）的地址（IP 或 IP 与 CS）压入堆栈，然后将新的目标地址（子程序或过程的首地址）装入 IP 或 IP 与 CS，于是控制转移到被调用的子程序。当调用结束时，指令 RET 从堆栈栈顶弹出之前压入的断点地址，重新装入 IP 或 IP 与 CS，从而返回到 CALL 的下一条指令（断点）处，继续运行。

（1）调用指令 CALL

指令格式：CALL　　OPRD

CALL 指令用于调用子程序（或过程）OPRD，与 JMP 指令类似，既可实现段内直接或间接调用，也可实现段间直接或间接调用，对应 4 种指令机器代码。直接调用时，目标地址信息就在 CALL 指令中；间接调用时，目标地址在由指令指定的寄存器或内存单元中。

若主程序与被调用的子程序在同一个代码段内，则是段内调用，此时子程序名包含属性 NEAR；若主程序与被调用的子程序不在同一个代码段内，则是段间调用，此时子程序名包含属性 FAR。汇编程序根据不同的子程序属性形成不同的目标地址并装入 IP 或 IP 与 CS。

① 段内或段间直接调用　CALL　　proc_name

功能：无条件调用子程序 proc_name。

若子程序 proc_name 属性是 NEAR，CALL 指令首先将断点的 IP 值入栈保护，再将 proc_name 提供的目标地址送入 IP，实现段内直接调用。

若子程序 proc_name 属性是 FAR，CALL 指令首先将当前断点的 CS、IP 值入栈保护，再将 proc_name 提供的目标地址送入 IP、CS，实现段间直接调用。

電子工業出版社

② 段内或段间间接调用　CALL　　reg/mem

功能：无条件调用由寄存器内容指定的子程序，或由存储器寻址方式提供的存储单元内容所指定的子程序。

若子程序属性是 NEAR，CALL 指令首先将断点的 IP 值入栈保护，再将 reg 或 mem 指定的子程序地址送入 IP，实现段内间接调用。

若子程序属性是 FAR，CALL 指令首先将当前断点的 CS、IP 值入栈保护，再将 mem 指定的子程序地址送入 IP、CS，实现段间间接调用。例如：

```
CALL    SUB_PROC1          ; SUB_PROC1 为 NEAR，段内直接调用。
CALL    3000H              ; 段内直接调用，3000H→IP
CALL    SUB_PROC2          ; SUB_PROC2 为 FAR，段间直接调用。
CALL    2000:3000H         ; 段间直接调用，3000H→IP，2000H→CS
CALL    AX                 ; 段内间接调用，目标地址(IP)由 AX 给出。
CALL    DWORD PTR [DI]     ; 段间间接调用，目标地址在[DI]、[DI+1]、[DI+2]、[DI+3]
                           ; 4 个内存单元中，前 2 字节为 IP，后 2 字节为 CS。
```

（2）返回指令 RET

指令格式：RET　或　RET　imm16

返回指令通常作为子程序的最后一条指令，用以返回到调用这个子程序的断点处。

① 若为段内（子程序被定义为 NEAR）返回，RET 指令把堆栈栈顶的一个字弹出至 IP，恢复断点处的偏移地址；若为段间（子程序被定义为 FAR）返回，则除了先弹出一个字至 IP，还要从当前栈顶继续弹出一个字到 CS，恢复断点处的段地址。

段内与段间返回的指令形式是一样的，但指令的机器代码不同，对应段内和段间两种返回情况。

② RET　imm16 指令中的 16 位立即数称为弹出（POP）值。若在 RET 指令中规定了弹出值，则 RET 指令在完成①所述的操作后，还需实现 SP←SP+imm16，即放弃堆栈中前 imm16 字节的内容。这个性能允许废除一些在执行 CALL 指令之前入栈的参数。

例如，下列程序段表示，由于在 RET 指令中规定了弹出值，从子程序返回后栈顶位置恢复到参数 N1、N2 入栈前的状态，N1、N2 被从堆栈中废除。

```
MOV     AX，N1    ; 主程序          PROG_A  PROC  NEAR        ; 子程序 A
PUSH    AX                                  …
MOV     AX，N2                              RET   4
PUSH    AX                        PROC_A    ENDP
CALL    PROG_A
MOV     SUM，AX
```

在主程序调用 PROG_A 子程序前，通过两次压栈将参数 N1 和 N2 传递给子程序使用，若子程序执行完毕后这两个参数不再有用，必须将它们废弃，否则每次执行该程序段后都会使堆栈空间损失 4 个单元。采用带弹出值的 RET 指令可以实现这个目的。

3．条件转移类指令

指令格式：Jcc　　OPRD

功能：根据 CPU 中一个或数个标志位状态所提供的转移条件 cc 决定程序的执行流向。若条件 cc 成立，则程序转移到指定目标标号 OPRD 的对应地址；否则顺序执行 Jcc 的下一

電子工業出版社

条指令。这类指令都是短转移（转移范围在距 Jcc 下一条指令地址的−128～+127 字节之间），采用相对转移形式。

所有条件转移指令对标志位无影响，指令的助记符也不唯一，如表 3-11 所示。

在程序中，条件转移类指令一般跟在算术/逻辑运算指令后面，通过检测运算结果所置的一个标志位的状态或者数个标志位的逻辑运算结果，来判断转移条件是否满足。归纳起来，这类指令主要分为 3 类。

表 3-11　条件转移类指令

指 令 功 能			指令书写格式（助记符）		测试条件
等于/结果为零		转移	JE/JZ	目标标号	ZF=1
不等于/结果不为零		转移	JNE/JNZ	目标标号	ZF=0
有进位/有借位		转移	JC	目标标号	CF=1
无进位/无借位		转移	JNC	目标标号	CF=0
溢出		转移	JO	目标标号	OF=1
不溢出		转移	JNO	目标标号	OF=0
奇偶性为 1/为偶		转移	JP/JPE	目标标号	PF=1
奇偶性为 0/为奇		转移	JNP/JPO	目标标号	PF=0
符号位为 1		转移	JS	目标标号	SF=1
符号位为 0		转移	JNS	目标标号	SF=0
对无符号数	高于/不低于也不等于	转移	JA/JNBE	目标标号	CF=0 且 ZF=0
	高于或等于/不低于	转移	JAE/JNB	目标标号	CF=0 或 ZF=1
	低于/不高于也不等于	转移	JB/JNAE	目标标号	CF=1 且 ZF=0
	低于或等于/不高于	转移	JBE/JNA	目标标号	CF=1 或 ZF=1
对带符号数	大于/不小于也不等于	转移	JG/JNLE	目标标号	SF⊕OF=0 且 ZF=0
	大于或等于/不小于	转移	JGE/JNL	目标标号	SF⊕OF=0 或 ZF=1
	小于/不大于也不等于	转移	JL/JNGE	目标标号	SF⊕OF=1 且 ZF=0
	小于或等于/不大于	转移	JLE/JNG	目标标号	SF⊕OF=1 或 ZF=1

（1）检测单个标志位状态的条件转移指令

表 3-11 中的前 10 条指令（5 个标志，10 条指令），适合根据某单个标志位的状态来决定是否转移的场合。例如：

```
ADD   AX，BX
JC    LAB1          ；当（AX+BX）有进位时，转至 LAB1。
CMP   CX，DX
JE    LAB2          ；当 CX=DX（即 CX－DX=0）时转至 LAB2，JE 也可用 JZ 替代。
```

（2）用于无符号数比较的条件转移指令

表 3-11 中的第 11～14 条指令，在执行时检测的是无符号数比较结果的特征标志位 CF 和 ZF，适合根据无符号数比较（相减）结果决定是否转移的场合。

（3）用于有符号数比较的条件转移指令

表 3-11 中的最后 4 条指令检测的是 SF、OF、ZF 标志位，这些标志位的逻辑组合可以表示两个有符号数间的大小关系，适合根据有符号数比较结果决定程序是否转移的情况。

【例 3-7】 01H 和 FEH 这两个数，若作为无符号数，01H 比 FEH 小；但作为有符号数，

電子工業出版社

01H 比 FEH（–2）大。当执行下列指令后，有 AL=01H，CF=1，OF=0，SF=0，ZF=0。

```
        MOV   AL, 01H
        CMP   AL, 0FEH
```

若要求 AL 中存放的无符号数大于 FEH 时转移，随后应使用“高于”指令 JA，这时因 AL 中的数小，程序并不发生转移（转移条件 CF=0 且 ZF=0 不满足）；若要求 AL 中存放的有符号数大于 FEH 转移，则应使用“大于”指令 JG，此时程序发生了转移，因为 AL 中的是大数（转移条件（SF⊕OF）=0 且 ZF=0 满足）。

注意：当测定两个无符号数或两个有符号数是否相等时，均可以使用指令 JE/JZ 或 JNE/JNZ。

条件转移指令有丰富的助记符，只要正确理解这些助记符的含义，可以不必关心发生转移时的具体标志位条件，编程时，只要按程序的意图选用相应助记符的条件转移指令即可。

【例 3-8】 将 AX 与 BX 内容相加。若结果为 0，则将 AX 清 0；若结果溢出，则程序返回，否则按顺序执行。

```
        ADD     AX, BX
        JZ      ZERO                ; 若加法结果为 0，转移至 ZERO
        JO      EXIT1               ; 若加法结果溢出，转移至 EXIT1
        …
ZERO:   MOV     AX, 0
EXIT1:  RET
```

【例 3-9】 设有 100H 字节的有符号数据存放在数据段中自 2000H 为起始地址的存储单元，编写程序找出其中的最小值并存入 2100H 单元。

```
MIN:    MOV     BX, 2000H       ; 取数据区首地址
        MOV     AL, [BX]        ; 取第一个数作为当前最小值
        MOV     CX, 0FFH        ; 置比较次数为 100H-1
LP1:    INC     BX
        CMP     AL, [BX]        ; 与数据区中数据比较
        JLE     LP2             ; 若 AL 小，不替换
        MOV     AL, [BX]        ; 替换 AL 中的最小值
LP2:    DEC     CX              ; 循环次数-1
        JNZ     LP1             ; 未比较完，继续
        MOV     [2100H], AL     ; 保存最小值
        HLT
```

4．循环控制指令

指令格式：LOOP　　OPRD
　　LOOPE/LOOPZ　　OPRD
　　LOOPNE/LOOPNZ　OPRD
　　JCXZ　　OPRD

功能：控制程序有规律地循环，它们默认 CX 为递减计数器来控制循环次数，根据对 CX 内容的测试结果决定程序是循环至目标地址 OPRD，还是顺序执行下一条指令。由此看来，循环控制指令类似条件转移指令，也是按条件来决定程序的走向，且目标地址 OPRD 在其下一条指令地址的−128～+127 字节范围内（短转移）。所有指令对标志位无影响。

循环控制指令的功能如表 3-12 所示。除了 JCXZ 指令，其余指令执行时都是先使 CX

電子工業出版社

寄存器内容减 1，然后依据 CX 中循环计数值是否为 0 来决定是否终止循环。

表 3-12　循环控制指令

指令功能	指令书写格式（助记符）		测试条件	
循环	LOOP	目标标号	CX←CX-1	CX≠0
相等/结果为 0，则循环	LOOPE/LOOPZ	目标标号	CX←CX-1	CX≠0 且 ZF=1
不等于/结果不为 0，则循环	LOOPNE/LOOPNZ	目标标号	CX←CX-1	CX≠0 且 ZF=0
CX=0，则转移	JCXZ	目标标号		CX=0

注意：LOOPE/LOOPZ 指令使用复合测试条件。该指令使 CX−1→CX，若 CX≠0 且 ZF=1（测试条件成立），则循环转移至目标标号，否则（CX=0 或 ZF=0）顺序执行其下面的指令。

LOOPNE/LOOPNZ 同 LOOPE/LOOPZ 指令类似，只不过转移条件是 CX≠0 且 ZF=0。

JCXZ 指令不影响 CX 的内容，仅在 CX=0 时控制转移到目标标号，否则顺序执行 JCXZ 的下一条指令。

【例 3-10】 用循环控制指令实现软件延时。

```
        MOV     CX, 0F000H          ; 预置循环次数初值
NEXT:   NOP                         ; 空操作（NOP）F000H 次，产生延时
        LOOP    NEXT
```

【例 3-11】设内存中 TAB1 处开始存有 100 个字符，编程查找$字符，若查到有该字符，将 TAB2 单元中存入 Y 字符，否则将 TAB2 单元中存入 N 字符。

```
        LEA     BX, TAB1        ; 设置数据指针
        MOV     CX, 100         ; 设置循环次数
LP0:    MOV     AL, [BX]        ; 取一个数
        CMP     AL, '$'         ; 比较是否为'$'
        JZ      LP1             ; 是，转移到 LP1，结束查找
        INC     BX              ; 不是，指针指向下一个数据
        LOOP    LP0             ; 未查完，转至 LP0 继续查找；查完，则结束循环
        MOV     TAB2, 'N'       ; 未查到'$'，存入'N'
        JMP     LP2
LP1:    MOV     TAB2, 'Y'       ; 查到，存入'Y'
LP2:    HLT
```

【例 3-12】 检查当前数据段所在的 64 KB 内存是否能够正确地进行读/写操作。可以先向每字节写入数据 01010101B（或 10101010B），然后读出来进行比较。若正确，则转至目标标号 OK 的程序段，否则进入出错处理程序段 ERROR。程序如下：

```
        MOV     CX, 0000H       ; 置 64K 循环初值
        MOV     BX, 0000H       ; 置偏移地址初值 0000H
        MOV     AL, 01010101B
AGAIN:  MOV     [BX], AL        ; 写入检查值到内存单元
        INC     BX
        CMP     [BX-1], AL      ; 将写入的检查值读出比较，是否正确？
        LOOPZ   AGAIN           ; CX←CX-1，若 CX≠0 且 ZF=1（正确），则循环至 AGAIN
        JCXZ    OK              ; CX=0、ZF=1，64K 个单元都正确，则转至 OK
ERROR:  …                       ; CX≠0、ZF=0，出错处理
        …
```


電子工業出版社

```
OK:     ...                              ; 读/写正确
```

当程序执行到 JCXZ 指令时，可能有两种情况：① CX=0，表示所有单元读/写都正确；② CX≠0，但 ZF=0，表示检查到某一单元时，写入与读出不一致，即出错。于是 JCXZ 根据 CX 内容是否为 0（为 0 表示正确，否则表示出错）使程序发生分支，进入不同的程序段。

5．中断指令

与转移指令、子程序调用指令、循环控制指令一样，中断也是改变程序执行顺序的一种方法。当系统遇到某个中断源的时候，会中断当前正在运行的程序，进入中断服务程序。

中断源有外部中断和内部中断两类。外部中断指外部硬件设备在需要与 CPU 交换数据时，向 CPU 发送中断请求信号以打断 CPU 的当前工作，要求 CPU 先执行它的服务程序，然后再返回继续工作的一种中断方式。内部中断用功能上类似外部中断的操作来改变程序执行方向和调用一个内部中断服务程序。与中断有关的指令有三条，见表 3-10。

（1）中断指令 INT/INTO

指令格式：INT　　imm8

　　INTO

INT　imm8 产生一个由 8 位立即数指定中断号的内部中断。该指令首先将标志寄存器 FR 和断点地址（CS 和 IP）进栈，并将标志位 TF 和 IF 清 0，然后将指定中断号的中断服务程序入口地址送 CS 和 IP，则程序发生转移。中断服务程序的入口地址是以中断号 imm8 作为索引，查找中断向量表得来的。

80x86/Pentium 系统规定，外部与内部两类中断源最多有 256 个，每个中断源的中断号 imm8 对应 0～255 中的一个值，又称为中断类型码。除了系统已用的若干低序号的中断源，其他可由用户在程序中使用。

INTO 为溢出中断指令，实际上是内部中断指令 INT imm8 的特例，中断类型码隐含为 4。只有当某运算结果使 OF=1（产生溢出）时，INTO 才产生中断。

（2）中断返回指令 IRET

指令格式：IRET

不管是外部中断（硬件引起）还是内部中断（软件引起），处理中断的中断服务程序的最后一条指令必须是 IRET，以退出中断过程，返回到中断时的断点处。IRET 指令执行的操作是从堆栈中弹出原先进栈保护的 IP、CS 和 FR 的值，继续被中断的程序执行。

中断指令的执行过程与子程序的调用非常相似，计算机常利用它为用户提供硬件设备的驱动程序。IBM PC 系列微机中的基本输入/输出系统 ROM BIOS 和操作系统 MS-DOS 都提供了丰富的中断服务程序让程序员调用。

3.3.6　处理器控制类指令

处理器控制类指令可分为三组，如表 3-13 所示。它们都是无操作数指令，除了对指定的标志位进行操作，对别的标志位没有影响。

1．标志位操作指令

CLC/STC/CMC：CLC 和 STC 指令使进位标志 CF 置为 0 或 1，CMC 使标志 CF 取反。

電子工業出版社

表 3-13　处理器控制类指令

类　别	指令功能	指令书写格式（助记符）
标志位操作	进位标志置 1	STC
	进位标志复位	CLC
	进位标志取反	CMC
	方向标志置 1	STD
	方向标志复位	CLD
	中断标志置 1	STI
	中断标志复位	CLI
外同步	停机	HLT
	等待	WAIT
	封锁总线	LOCK
	交权	ESC
空操作	空操作	NOP

CLD/STD：将方向标志 DF 置为 0 或 1，以控制串操作指令执行时 SI 和 DI 是增量还是减量。

CLI/STI：将中断标志 IF 置为 0 或 1，以控制 CPU 是否响应可屏蔽中断 INTR 引线上出现的外部中断请求。CLI 禁止 CPU 响应（IF=0），STI 允许 CPU 响应（IF=1）。

2．外同步类指令

外同步类指令的作用是实现 CPU 与外部事件的同步，共 4 条。

（1）处理器暂停指令 HLT

HLT 指令使 CPU 进入暂停状态，不做任何操作，程序停止执行。只有当 CPU 复位或外部中断发生时，CPU 才能退出暂停状态。

对含有外部中断的系统进行程序调试时，常用 HLT 指令模拟主程序。当中断到来使 CPU 退出暂停而去执行中断服务程序时，当前的断点地址 CS:IP 指向 HLT 的下一条指令，在中断服务完成返回时，返回到此断点处继续程序的执行。

（2）处理器等待指令 WAIT

WAIT 指令在 CPU 的 $\overline{\text{TEST}}$ 引脚无效（高电平）时，使 CPU 进入等待状态。在等待状态，CPU 没有任何操作，直到出现外部中断为止。当外部中断发生时，进栈保护的 IP 值就是 WAIT 指令所在单元的地址。所以中断结束后又返回到等待状态，直到 $\overline{\text{TEST}}$ 引脚上信号有效（低电平），CPU 才退出等待，执行下一条指令。

（3）总线锁定前缀 LOCK

LOCK 是一个单字节的前缀，可以加在任何指令前面，使 CPU 在执行这条指令期间保持输出引脚 $\overline{\text{LOCK}}$ 有效（低电平），即将总线封锁使其他控制器不能控制总线，直到该指令执行完后，总线封锁解除。在多处理器系统中，可以用这个前缀来对共享资源进行控制。

（4）处理器交权指令 ESC

ESC 指令将浮点指令交给浮点运算协处理器执行。协处理器的运算指令是与 CPU 的指令组合在一个指令流中运行的，所有的协处理器运算指令的前几位编码，对于 CPU 来说就是一条 ESC 指令。当执行 ESC 指令时，80x86/Pentium CPU 就把控制权交给协处理器，以完成协处理器运算指令的操作。

3．空操作指令 NOP

NOP 指令的作用是使 CPU 完成一次空操作，仅使指令指针 IP 加 1，不做任何其他操作，空耗一个指令执行周期。与 HLT 指令不同，NOP 指令执行后，CPU 继续执行其后的指令。

3.4　80x86/Pentium CPU 指令系统

8086/8088 后每款新 CPU 都完全兼容老产品，即 80x86/Pentium 系列 CPU 的指令系统向下兼容，不仅包容了原有的 8 位/16 位基本指令集，还增强与增加了新的指令，全面支持 32 位数据类型、32 位操作数和 32 位寻址方式。在 80x86/Pentium 处理器的发展进程中，这

電子工業出版社

些扩充增加的指令极大地提升了 CPU 的性能，包括：32 位整数指令集、多媒体 MMX 指令集、数据流 SIMD 扩展 SSE 指令集，以及用于保护模式编程的系统控制类指令等。

3.4.1 80286 CPU 的增强与增加指令

80286 是超级 16 位 CPU，从它开始的 CPU 都将 8086/8088 的工作模式作为实模式，在此基础上增加了保护等其他工作模式。80286 CPU 增强与增加的指令如表 3-14 所示。

表 3-14 80286 CPU 增强与增加的指令

	类 别		指令书写格式（助记符）	说 明
增强	传送类（堆栈操作）		PUSH imm16	允许将字立即数压入堆栈，如果给出的数不足 16 位，指令会在自动扩展后压入堆栈
	算术运算类（有符号数乘法）		IMUL reg16, reg16	允许有两个或三个操作数： OPRD1←OPRD1×OPRD2（两个） OPRD1←OPRD2×OPRD3（三个） 乘积的长度限制与 OPRD1 的一致，如果溢出，则溢出部分丢掉，并置 CF=OF=1
			IMUL reg16, mem16	
			IMUL reg16, imm	
			IMUL reg16, reg16, imm	
			IMUL reg16, mem16, imm	
	逻辑运算与移位类（移位）		SHL reg/mem, imm8	允许立即数 imm8 在 1～31 之间
			其余 SAL、SAR、SHR、ROL、ROR、RCL、RCR 指令同 SHL	
增加	传送类（堆栈操作）		PUSHA	将所有通用寄存器的内容压入或弹出堆栈。压入顺序为 AX、CX、BX、DX、SP、BP、SI、DI，弹出顺序与压入顺序相反
			POPA	
	串操作类（从端口接收或发送数据串）		INS [ES:DI], DX	从 DX 指定的外设端口输入 1 字节或 1 个字到 ES:DI 指定的存储单元，或将[DS:SI]中的 1 字节或 1 个字输出到 DX 指定的外设端口，且根据 DF 和存储器操作数的属性修改 DI、SI 的值
			OUTS DX, [DS:SI]	
			INSB/ INSW，OUTB /OUTW	
			[REP] INS/ OUTS	
	高级语言类	数组边界检查	BOUND reg16，mem32	验证 reg16 中的操作数是否在 mem32 所指向的两个界限内
		建立堆栈空间	ENTER imm16，imm8	为局部变量建立一个堆栈区
		取消堆栈空间	LEAVE	撤销前面 ENTER 指令的动作

1．80286 CPU 的增强指令

在 8086/8088 指令系统中，许多指令不能用立即数作为源操作数，80286 增强了此功能，允许立即数作为源操作数，如：① 堆栈操作指令 PUSH 允许将 16 位立即数直接压栈；② 允许移位指令中移位计数值从原来规定的常数 1 扩充至在 1～31 之间的立即数。例如：

```
SAR     DX, 3
ROL     BYTE  PTR [BX], 5
```

特别地，有符号数乘法指令不但允许立即数作为一个乘数操作数，而且允许有两个或三个操作数。例如：

```
IMUL    BX, 50                ; BX←BX×50
IMUL    AX, [BX+DI], 0342H    ; AX←0342H×DS:[BX+DI]
IMUL    AX, BX, 20            ; AX←20×BX
```

2．80286 CPU 的增加指令

80286 CPU 增加的一些新功能指令如下。

① 16 位通用寄存器全部进栈/出栈指令 PUSHA/POPA。

② 从外设端口进行串输入的指令 INS/INSB/INSW 和串输出指令 OUTS/OUTSB/OUTSW，它们都可在其前面加重复前缀 REP，以连续实现整个串的输入/输出操作，这时

電子工業出版社

CX 寄存器中为重复操作的次数。例如，以下程序段从端口 PORT 输入 100 字节存到附加段中以 TABLE 为首地址的内存区。

```
        CLD                             ; DF=0，按地址增量传送
        LEA     DI, TABLE
        MOV     CX, 100                 ; CX 中为计数值
        MOV     DX, PORT
        REP     INSB                    ; [ES:DI]←[DX]，DI←DI+1，CX←CX-1，直至 CX=0
```

③ 高级语言类指令 BOUND、ENTER、LEAVE。BOUND 指令用于验证在指定寄存器 reg16 中的操作数是否在 mem32 所指向的两个界限内。若不在，则产生一个 5 号中断。指令中假定上、下界值（即数组的起始和结束地址）依次存放在相邻存储单元中。例如：

```
ARRAY   DW      0000H, 0063H            ; 定义数组的最小下标 0 及最大下标 99（63H）
NUMB    DW      0019H
        …
        MOV     BX, NUMB                ; BX 中为被测下标值 25（19H）
        BOUND   BX, ARRAY               ; 检查被测下标值是否在规定的下标边界范围之内
        …
```

程序中的每个子程序都有自己的局部变量，这些局部变量只有当它们所在的子程序执行时才有意义。为了保存这些局部变量，当执行到这些子程序时，应为它们建立相应的堆栈框架，退出子程序时再撤除。80286 用 ENTER、LEAVE 两条指令来完成这些功能。

ENTER 指令为局部变量建立一个堆栈区，指令中的目标操作数 imm16 指出子程序所要使用的堆栈字节数，源操作数 imm8 指出子程序的嵌套层数：0～31。LEAVE 指令用于撤销前面 ENTER 指令的动作。例如：

```
TASK    PROC NEAR
        ENTER   60, 4                   ; 建立堆栈区并保存 60 字节长的局部变量
        …
        LEAVE                           ; 撤销建立的栈空间，恢复栈指针
        RET
TASK    ENDP
```

3．新增的保护模式编程指令

保护模式与实模式的不同，主要表现在它对存储器的管理和使用上。在开机或复位后，80x86/Pentium CPU 先进入实地址模式。如果进入保护模式，需要对各系统表和寄存器进行初始化，包括建立全局/中断描述符表 GDT/IDT、任务状态段 TSS 以及设置 GDTR、CR0 等。切换进入保护模式后，系统需要重新对 CS、SS、DS、ES 及任务状态段寄存器 TR 等进行设置。80286 为保护模式增加了一套如表 3-15 所示的编程指令，又称为系统控制类指令，包括描述符表操作和任务切换两大类。

表 3-15　80286 CPU 保护模式编程指令

类别	指令书写格式（助记符）	操　作	指令功能
描述符表操作	LAR　reg16, reg16/mem16	reg16←reg16/mem16	将 reg16/mem16 中选择符指示的访问权限装入目标 reg16 的高字节，低字节置 0
	LSL　reg16, reg16/mem16	reg16←reg16/mem16	将 reg16/mem16 中选择符指示的描述符的段界限装入目标 reg16
	LGDT　mem48	GDTR←mem48	将 6 字节的 mem48 装入全局描述符表寄存器
	SGDT　mem48	mem48←GDTR	将全局描述符表寄存器存入 6 字节的 mem48
	LIDT　mem48	IDTR←mem48	将 6 字节的 mem48 装入中断描述符表寄存器
	SIDT　mem48	mem48←IDTR	将中断描述符表寄存器存入 6 字节的 mem48
	LLDT　reg16/mem16	LDTR←reg16/mem16	将 reg16/mem16 装入局部描述符表寄存器
	SLDT　reg16/mem16	reg16/mem16←LDTR	将局部描述符表寄存器存入 reg16/mem16

電子工業出版社

续表

类别	指令书写格式（助记符）	操　作	指令功能
任务切换指令	LTR　reg16/mem16	TR←reg16/mem16	将 reg16/mem16 装入任务状态段寄存器
	STR　reg16/mem16	reg16/mem16←TR	将任务状态段寄存器内容存入 reg16/mem16
	LMSW　reg16/mem16	MSW←reg16/mem16	将 reg16/mem16 装入机器状态字 MSW（CR_0 的低 16 位）
	SMSW　reg16/mem16	reg16/mem16←MSW	将机器状态字 MSW 存入 reg16/mem16
	VERR　reg16/mem16		验证当前特权级是否对 reg16/mem16 中的段选择符所指定的段进行读操作，是则 ZF=1
	VERW　reg16/mem16		验证当前特权级是否对 reg16/mem16 中的段选择符所指定的段进行写操作，是则 ZF=1
	ARPL　reg16/mem16, reg16		使目标操作数指定的段选择符的请求特权级 RPL 与源操作数的 RPL 匹配
	CLTS		清除机器状态字 MSW（CR_0 的低 16 位）中的任务切换标志位

3.4.2　80386 CPU 的增强与增加指令

80386 是 32 位 CPU，兼容全部的 80286 指令，且直接扩展为 32 位，表现在：① 所有指令都扩展支持 32 位操作数，包括 32 位立即数；② 所有涉及存储器寻址的指令都可以使用 32 位寻址方式。MASM 5.0 开始支持 80386 指令系统。

80386 增强与增加的指令如表 3-16 所示。增强的指令都是允许 32 位操作后出现的新的指令助记符；增加的指令中，控制、调试与测试寄存器传送指令属于系统控制类指令。

表 3-16　80386 CPU 增强与增加的指令

	类　别	指令书写格式（助记符）	指令功能
增强	传送类	PUSHAD/POPAD	将通用寄存器 EAX、ECX、EBX、EDX、ESP、EBP、ESI、EDI 顺序压入堆栈，弹出顺序与压入顺序相反
		PUSHFD/POPFD	将 32 位标志寄存器 EFLAGS 进栈/出栈
		LFS/LGS　reg, mem	将 mem 中存放的偏移地址送 16 或 32 位 reg，段地址送 FS/GS
	算术运算类	CWDE	将 AX 中 16 位有符号数扩展为 32 位，存放在 EAX 中
		CDQ	将 EAX 中 32 位有符号数扩展为 64 位，存放在 EDX :EAX 中
	串操作类	所有串操作指令后面扩展 D，如：MOVSD、OUTSD…	将所有串操作直接扩充至 32 位
增加	传送类	MOVSX/MOVZX　reg, reg/mem	若两个有（无）符号数等长，则进行传送操作；否则，扩展补齐后传送
	逻辑运算与移位类	SHLD/SHRD　reg/mem, reg, CL	reg/mem 和 reg 一起左（右）移 CL 位或 imm8（1～31）位，且 reg 不变，移出位→CF
		SHLD/SHRD　reg/mem, reg, imm8	
	位操作类	BT　reg/mem, reg/imm8	将目标操作数中由源操作数指定的位送 CF
		BTR/BTS/BTC　reg/mem, reg/imm8	将目标操作数中由源操作数指定的位送 CF，然后将该位清 0、置 1、取反
		BSF/BSR　reg, reg/mem	将 reg/mem 从低位（高位）向高位（低位）逐位扫描，找出第一个 1，将其位号存入 reg，且 ZF=0，否则 ZF=1
	条件设置类	SETcc　reg8/mem8	测试条件 cc（为标志寄存器中状态之一）成立，则 reg8/mem8 置 1，否则 reg8/mem8 清 0
	控制、调试与测试寄存器传送指令	MOV　CR*n*, reg32	控制寄存器 CR*n*（*n*=0～4）与 32 位通用寄存器间的传送
		MOV　reg32, CR*n*	
		MOV　DR*n*, reg32	调试寄存器 DR*n*（*n*=0～7）与 32 位通用寄存器间的传送
		MOV　reg32, DR*n*	
		MOV　TR*n*, reg32	测试寄存器 TR*n*（*n*=3～7）与 32 位通用寄存器间的传送

電子工業出版社

对 80386 新增指令的使用举例如下。

【例 3-13】 传送指令 MOVSX/MOVZX 可以对有符号数/无符号数进行扩展传送。

```
MOVSX    ECX, AL          ; 将 AL 的内容带符号扩展为 32 位送入 ECX
MOVZX    EAX, CX          ; 将 CX 中 16 位数加 0 扩展为 32 位送入 ECX
```

【例 3-14】 双精度左移/右移指令 SHLD/SHRD 是三操作数指令，将 OPRD1 和 OPRD2 两个 16 位或 32 位操作数连接成双精度值（32 位或 64 位），然后向左或向右移位 OPRD3 次。移位时，OPRD2 的内容移入 OPRD1，而 OPRD2 本身不变。CF 中为 OPRD1 移出的最后一位。

```
MOV      AX, 2A80H
MOV      BX, 9A78H
SHLD     AX, BX, 8              ; AX=809AH，BX=9A78H，CF=0。
```

【例 3-15】位测试指令 BT/BTR/BTS/BTC 测试目标操作数 OPRD1 中由源操作数 OPRD2 指定的那一位（最低位为 b_0），并将测试位送入 CF。测试完成后，BT 指令不改变目标操作数，其他指令则分别将测试位清 0、置 1 或取反。

```
MOV      CX, 4
BT       [BX], CX               ; 检查[BX]的 b4 位，且将 b4 位放入 CF 中
JC       NEXT                   ; b4 位=1 转移至 NEXT
```

【例 3-16】 位扫描指令 BSF/BSR 分别实现前向位扫描和后向位扫描，当对 16 位或 32 位源操作数 OPRD2 从低位 b_0 到高位（b_{15} 或 b_{31}）扫描或反向扫描时，将扫描到的第一个 1 的位号送入 OPRD1，并将 ZF 清 0 或置 1。

```
MOV      BX，40A0H
BSF      AX，BX                 ; 指令执行后，AX=5
BSR      AX，BX                 ; 指令执行后，AX=14
```

【例 3-17】 条件设置指令 SETcc　OPRD 有 16 条，根据指令中给出的条件“cc”是否满足来设置 OPRD 指定的 8 位寄存器或存储器操作数，常用于需要判断转移的场合，但无须转移指令，如表 3-17 所示。

表 3-17　条件设置指令表

指令书写格式（助记符）	设置条件	指令条件说明
SETO　reg8/mem8	OF=1	溢出
SETNO　reg8/mem8	OF=0	无溢出
SETC/SETB/SETNAE　reg8/mem8	CF=1	有进位/低于/不高于或等于
SETNC/SETNB/SETAE　reg8/mem8	CF=0	无进位/不低于/高于或等于
SETZ/SETE　reg8/mem8	ZF=1	为零/等于
SETNZ/SETNE　reg8/mem8	ZF=0	非零/不等于
SETS　reg8/mem8	SF=1	为负数
SETNS　reg8/mem8	SF=0	为正数
SETP/SETPE　reg8/mem8	PF=1	检验为偶
SETNP/SETPO　reg8/mem8	PF=0	检验为奇
SETA/SETNBE　reg8/mem8	CF=0 且 ZF=0	高于/不低于或等于
SETNA/SETBE　reg8/mem8	CF=1 或 ZF=1	不高于/低于或等于
SETG/SETNLE　reg8/mem8	ZF=0 且 SF=OF	大于/不小于或等于
SETGE/SETNL　reg8/mem8	ZF=1 或 SF=OF	大于或等于/不小于
SETL/SETNGE　reg8/mem8	ZF=0 且 SF≠OF	小于/不大于或等于
SETLE/SETNG　reg8/mem8	ZF=1 或 SF≠OF	小于或等于/不大于

電子工業出版社

在32位CPU中增加了一些系统控制用的寄存器，包括：控制寄存器CRn、调试寄存器DRn和测试寄存器TRn，所以增加了相关传送指令对它们进行操作，如表3-16中的控制、调试与测试寄存器传送指令。这些指令一般只在系统程序中使用，且在Pentium CPU以后才支持CR4。

3.4.3 80486 CPU 增加的指令

80486 CPU 在 80386 体系结构上进行了一些扩展，因而只有新增指令，如表 3-18 所示。其中，三条管理 Cache 的指令属于系统控制类指令。MASM 6.0 开始支持 80486 指令系统。

表 3-18 80486 CPU 增加的指令

类 别	指令书写格式（助记符）	指令功能
传送类	BSWAP reg32	将 32 位寄存器中的最高和最低字节交换，中间 2 字节交换
算术运算类	XADD reg/mem, reg	reg/mem←reg/mem+reg，且 reg←reg/mem 中原来的值
	CMPXCHG reg/mem, reg	若 reg/mem 与 AL 或 AX 或 EAX 比较相等，则 reg/mem←reg，且 ZF=1；否则，AL/AX/EAX←reg/mem，且 ZF=0
Cache 管理类	INVD	将内部和外部 Cache 的内容作废，外部 Cache 的数据不写回内存
	WBINVD	先清除内部 Cache，然后将外部 Cache 的内容写回内存，再清除外部 Cache
	INVLPG	使页式管理机构内的高速缓冲器 TLB 中的某项作废

3.4.4 Pentium 系列 CPU 增加的指令

Pentium 系列 CPU 主要包括：Pentium、Pentium MMX、Pentium III、Pentium 4。作为现阶段的主流体系结构，其指令系统最主要的是增强了多媒体、图形图像和 Internet 处理能力。

1. Pentium CPU 的指令系统

Pentium CPU 的指令系统主要是在 80486 指令集上增加了三条整数指令和三条系统控制指令，如表 3-19 所示。MASM 6.11 开始支持 Pentium CPU 指令系统。

表 3-19 Pentium CPU 增加的指令

类 别		指令书写格式（助记符）	指令功能
新增整数指令	比较和交换	CMPXCHG8B mem64	若 EDX:EAX=mem64，则 ZF←1，mem64←ECX:EBX；否则，ZF=0，EDX:EAX←mem64
	读 CPU 标识	CPUID	若 EAX=0，EAX:EBX:ECX:EDX←厂商信息；若 EAX=1，EAX:EBX:ECX:EDX←CPU 信息
	读时间标记计数器	RDTSC	将 Pentium 处理器中 64 位时间标记计数器 TSC（每个时钟周期自动加 1）读到 EDX:EAX 中
新增系统控制指令	读专用模式寄存器	RDMSR	将 ECX 中（00H～13H）指明的专用模式寄存器 MSR（Model-Specific Registers）中的 64 位数据读到 EDX:EAX 中
	写专用模式寄存器	WRMSR	将 EDX:EAX 中的 64 位数据写入 ECX 中（00H～13H）指明的专用模式寄存器 MSR 中
	由系统管理模式返回	RSM	使 Pentium CPU 从系统管理模式（SMM）返回到原来的实模式或保护模式

简单说明：从 Pentium CPU 开始，可以通过处理器识别指令 CPUID 获得 CPU 的 ID，以便程序在合适的 CPU 上运行。系统上电时 64 位的时间标记计数器 TSC 清 0，通过读 TSC 可检测程序运行速度。专用模式寄存器 MSR 一般由系统开发人员用于跟踪程序、查错等。

電子工業出版社

2．Pentium MMX CPU 的指令系统

Pentium MMX CPU 在 Pentium 指令系统的基础上，增加了 MMX（MuliMedia eXtension，多媒体扩展）指令集，这是 Intel 公司在推出 Pentium MMX CPU 时公布的一项针对多媒体数据特点的增强技术，包含 7 类共 57 条多媒体处理专用指令，如表 3-20 所示。除了传送和清除指令，MMX 指令集的助记符都以 P 开头，表征其处理的是“紧缩（Packed）型整型数据”。即 MMX 指令处理的一个 64 位数据是由多个 8、16、32 位的整型数据组合而成的。

表 3-20　MMX 指令集

类　别	指令书写格式（助记符）		指令功能
数据传送（2 条）	MOVD	mm, reg/mem32	将 32 位数据从寄存器、内存中移入或移出 MMX 寄存器 mm
	MOVD	reg/mem32, mm	
	MOVQ	mm, reg/mem64	将 64 位数据从寄存器、内存中移入或移出 MMX 寄存器 mm
	MOVQ	reg/mem64, mm	
算术运算（17 条）	PADDB/PSUBB	mm, mm/mem64	按字节（B）、字（W）、双字（D）数据包进行加法/减法运算，各独立单位不产生进位
	PADDW/PSUBW	mm, mm/mem64	
	PADDD/PSUBD	mm, mm/mem64	
	PADDSB/PSUBSB	mm, mm/mem64	带符号的字节、字数据包进行饱和加法、减法运算
	PADDSW/PSUBSW	mm, mm/mem64	
	PADDUSB/PSUBUSB	mm, mm/mem64	无符号的字节、字数据包进行饱和加法、减法运算
	PADDUSW/PSUBUSW	mm, mm/mem64	
	PMULHW/PMULLW	mm, mm/mem64	对应的四字分别相乘后取 4 个乘积高、低 16 位，结果为四字
	PMADDWD	mm, mm/mem64	对应的四字分别相乘后将结果两两相加，操作结果为 2 个双字
比较操作（6 条）	PCMPEQUB/ PCMPGTB	mm, mm/mem64	对字节包、字包、双字包数据做等于、大于比较，比较结果为“真”或“假”的逻辑值
	PCMPEQUW/ PCMPGTW	mm, mm/mem64	
	PCMPEQUD/ PCMPGTD	mm, mm/mem64	
逻辑运算（4 条）	PAND/PANDN/POR/PXOR	mm, mm/mem64	按位进行与、与非、或、异或操作
移位指令（18 条）	PSLLW/ PSLLD/ PSLLQ	mm, mm/mem64/imm8	将目标操作数以字、双字、四字为单位逻辑左、右移，移位次数由源操作数指定
	PSRLW/ PSRLD/ PSRLQ	mm, mm/mem64/imm8	
	PSRAW/ PSRAD/ PSRAQ	mm, mm/mem64/imm8	将目标操作数以字、双字、四字为单位算术右移，移位次数由源操作数指定
转换指令（9 条）	PACKUSWB	mm, mm/mem64	无符号字包压缩成字节包
	PACKSSWB/ PACKSSDW	mm, mm/mem64	带符号字、双字包压缩成字节、字包
	PUNPCKHBW/PUNPCKHWD	mm, mm/mem64	分离源与目标操作数的高位字节、字、双字，重新交叉组合成字、双字、四字
	PUNPCKHDQ	mm, mm/mem64	
	PUNPCKLBW/PUNPCKLWD	mm, mm/mem64	分离源与目标操作数的低位字节/字/双字，重新交叉组合成字、双字、四字
	PUNPCKLDQ	mm, mm/mem64	
MMX 状态管理（1 条）	EMMS		清除 MMX 状态，使后续浮点指令可以使用浮点寄存器

注：mm 指 1 个 64 位 MMX 寄存器 MM7～MM0；B—紧缩字节；W—紧缩字；D—紧缩双字；Q—紧缩 4 字。
饱和运算的含义：当运算结果发生溢出时，其结果值就用最大值（加法）或最小值（减法）代替。

多媒体数据的每个样本点由多个数据组成，而且呈现出某种对称性，如音频信号中的左、右两个声道，视频信号中的 R、G、B 三基色信号，3D（三维）图形处理中的 *X*、*Y*、*Z* 坐标数据等，Pentium MMX 之前的“单指令单数据”指令系统处理这样的数据效率是非常有限的。

MMX 技术的主要特点是，提出了“单指令多数据（Single Instruction Multiple Data，

電子工業出版社

SIMD）”的概念，使得一条指令能够同时处理多个数据，即一条指令的操作数中包含有多个经过组合的紧缩数据。MMX 的这种思想将 CPU 处理数据的速度从一次处理一个变成了一次处理一组，这极大地提高了 Pentium/Pentium Pro CPU 处理多媒体数据的性能。

紧缩整型数据分为 4 种数据类型：紧缩字节、紧缩字、紧缩双字、紧缩四字。

- 紧缩字节（Packed Byte）：8 字节组合成为一个 64 位数据。
- 紧缩字（Packed Word）：4 个字组合成为一个 64 位数据。
- 紧缩双字（Packed Doubleword）：2 个双字组合成为一个 64 位数据。
- 紧缩四字（Packed Quadword）：一个 64 位数据。

MMX 技术使用 8 个 64 位的 MMX 寄存器来适应多个数据的并行处理，它们以 MM7～MM0 来标识，与浮点处理单元 FPU 的 8 个 80 位寄存器共享，使用了其中的 64 位尾数部分。因此，MMX 指令集与 x87 浮点运算指令不能同时执行，两者之间只能分时切换，这在需密集切换时无疑会造成整个系统运行质量的下降。

3．Pentium III CPU 的指令系统

MMX 技术只适合整型数据处理，不能用于浮点处理。为此，Intel 公司使用 MMX 指令的 SIMD 关键技术，在 Pentium III 中增加了适应网络发展的数据流扩展（Streaming SIMD Extension，SSE）技术，新增 SSE 指令 70 条，使得浮点数处理具备了单指令多数据的能力。

为了解决与浮点处理单元 FPU 的冲突问题，SSE 技术新增了 8 个 128 位的 SIMD 浮点寄存器 XMM7～XMM0，每个寄存器可存放 4 个 32 位的单精度浮点数，同时处理的数据比 64 位的 MMX 翻了一番。应该说，MMX 指令集开创了 Intel CPU 引入 SIMD 指令集之先河，MMX 可以同时对 2 个 32 位的整数进行操作，而 SSE 可以同时对 4 个 32 位的浮点数操作，并避开了 MMX 中使用浮点寄存器的冲突。SSE 对于 3D 几何运算，动画处理，图像处理，音频/视频信号的编辑、合成、识别以及解压缩处理的性能都有极为明显的改善。

Pentium III 中 SSE 指令集的 70 条指令分为如下 3 类。

（1）提高 3D 图形运算效率的 50 条单指令多数据浮点运算指令

计算机表示的三维图形是用大量的表示图形顶点的浮点数组成的，通过操作这些顶点数据就可以改变三维图形的外观。通过使用 SSE 指令集及 128 位的浮点寄存器，CPU 可以在一个时钟周期内处理更多的数据，可以显著改善一些在三维操作中经常用到的计算，如矩阵变换、矩阵的加、减、乘运算以及光照计算等，大大加快了三维图形的顶点计算速度，结合复杂的算法可以创造出更为生动的三维图像效果。

（2）12 条 MMX 整数运算增强指令

Pentium III 采用了比 Pentium II 更先进的算法，进一步增强了 CPU 在视频和图形处理方面的能力。

（3）8 条优化内存中连续数据块传输的指令

Pentium III 采用新的数据预存取技术，减少了 CPU 处理连续数据流的中间环节，提高了连续数据流的处理效率，使视频、音频处理的速度更快、质量更高。

4．Pentium 4 CPU 的指令系统

MMX 与 SSE 指令集主要提供了整型数据及单精度浮点数据的并行处理能力。

2000 年年底，Intel 公司开始推出 Pentium 4 CPU，从 Willamette 核心的 Pentium 4 开始采用 SIMD 技术，基于 SSE 开发了第 2 代浮点多媒体处理器指令集 SSE2，扩展了双精度浮

電子工業出版社

点数并行处理能力。

SSE2 指令共 144 条，通过支持多种数据类型（如双字和四字）的算术运算，使最初的 SSE 指令得到增强，支持灵活且动态范围更广的计算功能。同时，随 MMX 技术引进的 SIMD 整数指令从 64 位扩展到了 128 位，使 SIMD 整数类型操作的有效执行率成倍提高。最具特点的是，SSE2 中增加了 64 位双精度浮点 SIMD 指令，允许以 SIMD 格式同时执行两个浮点操作，提供双倍精度操作支持；加上对 Cache 的控制指令，使得 SSE2 非常适合 3D 渲染、图形驱动、游戏和多媒体编码等应用。

2004 年，Intel 公司发布了 Prescott 核心的 Pentium 4 处理器，SSE 家族又添新丁 SSE3。SSE3 只有 13 条指令，其中 10 条用于完善 MMX、SSE、SSE2 指令，1 条用于 x87 浮点处理单元编程中浮点数转换为整数的加速，另外 2 条用于加速线程的同步。这些新增指令强化了 CPU 在浮点数到整数的转换（不需要进行运算模式切换）、复杂算法（DSP 及 3D）、视频编码、SIMD 浮点寄存器操作以及线程同步等 5 方面的表现，达到提升多媒体和游戏性能的目的。

经过多年的发展，多媒体指令集已经成为 CPU 密不可分的一部分，后继扩展了 SSE3 的产品 SSSE3 和 SSE4。SSSE3（Supplemental Streaming SIMD Extensions 3）是内置于 Intel 微处理器中的与多媒体关联的 16 条指令集，于 2006 年 7 月首次装载在 65nm Core 2 Duo 处理器中，进一步增强了 CPU 的多媒体、图形、图像和 Internet 等处理能力。SSE4 指令集被认为是 2001 年以来 Intel 微处理器最重要的指令集扩展，包含两个分支：SSE4.1 + SSE4.2，共 54 条指令。Intel Penryn 处理器中加入了对 SSE4.1 的支持，增加了 47 条新指令，其中包括使浮点运算专门化的 6 条浮点型点积运算指令，支持单精度、双精度浮点运算及浮点产生操作，提升了处理器在图形、三维图像与游戏、视频编码与影音处理等方面的性能表现。Nehalem 处理器中，相对于 SSE4.1，后来加入了 7 条新指令，完整地实现了 SSE4 指令集。

3.5 80x87 浮点运算指令

实际上，80x86/Pentium 系列 CPU 的指令系统分为整数指令集（80x86 指令）和浮点运算指令集（80x87 指令）两大部分。独立的浮点运算指令集专门用于浮点处理单元 FPU（80x87），分为传送、算术运算、超越函数、比较、FPU 控制等 5 类。

3.5.1 80x87 的数据类型与格式

Intel 80x87 FPU 支持如下 3 类（整型、BCD、浮点）7 种数据类型。

- 单精度浮点数（32 位短实数）——由 1 位符号、8 位指数、23 位有效数组成。
- 双精度浮点数（64 位长实数）——由 1 位符号、11 位指数、52 位有效数组成。
- 扩展精度浮点数（80 位临时实数）——由 1 位符号、15 位指数、64 位有效数组成。
- 很多计算机中并没有 80 位扩展精度数据类型，80x87 FPU 主要在内部使用它来存储中间结果，以保证最终数值的精度。
- 字整数（16 位）、短整数（32 位）、长整数（64 位）——三种整型数据的最高有效位都表示符号，对整数类型的支持可以使 16 位的 x86 CPU 利用 FPU 来完成 32 位或 64 位整数算术运算。

電子工業出版社

- BCD 码数（80 位）——占 10 字节，低端 9 字节为 18 个压缩 BCD 码数，最高字节的最高有效位为符号位，其余 7 位未定义。

3.5.2 浮点寄存器

浮点执行环境的寄存器主要是 8 个通用浮点数据寄存器 FPR_0～FPR_7 和几个专用寄存器，它们是状态寄存器、控制寄存器、标记寄存器等。

每个浮点寄存器（FPR_0～FPR_7）都是 80 位的，以扩展精度格式存储数据。当其他类型数据压入数据寄存器时，FPU 自动转换成扩展精度；相反，数据寄存器的数据取出时，系统也会自动转换成要求的数据类型。

8 个浮点数据寄存器组成首尾相接的堆栈，当前栈顶 ST(0)指向的 FPR_x 由状态寄存器中 TOP 字段指明。浮点数据寄存器不采用随机存取，而是按照“后进先出”的堆栈原则工作，并且首尾循环，常称其为浮点数据栈。

向浮点数据寄存器传送（Load）数据时是入栈，堆栈指针 TOP 先减 1，再将数据压入栈顶寄存器；从浮点数据寄存器取出（Store）数据时是出栈，先将栈顶寄存器数据弹出，再修改堆栈指针使 TOP 加 1。浮点数据栈首尾循环相接，若当前栈顶 TOP=0（即 ST(0) = FPR_0），那么，入栈操作后就使 TOP=7（即 ST(0) = FPR_7），数据被压入 FPR_7。

为了表明浮点数据寄存器中数据的性质，对应每个 FPR 寄存器都有一个 2 位的标记（tag_x）位，这 8 个标记 tag_0～tag_7 组成一个 16 位的标记寄存器。浮点状态寄存器表明 FPU 当前的各种操作状态，每条浮点指令都对它进行修改以反映执行结果。浮点控制寄存器用于 FPU 的异常屏蔽、精度控制和舍入操作。

3.5.3 80x87 指令简介

浮点指令归属于 ESC 指令，其前 5 位的操作码都是 11011B，指令助记符具有如表 3-21 所示的特征，以标示指令的类属。各类浮点指令功能如表 3-22 所示。

以下程序段描述了已知半径 R 计算圆面积的过程：

```
FINIT                    ; 第一条指令必须是 FINIT，初始化 FPU
FLDPI                    ; π 压入栈顶
FLD     F64D1            ; 半径值 R（双精度浮点数）压入栈顶
FMUL    ST(0), ST(0)     ; 乘积：R×R
FMUL                     ; 求面积 π×R², 并出栈
FSTP    F64D2            ; 将栈顶弹出存入 F64D2，为面积 π×R²
```

表 3-21 浮点指令助记符的特征

助记符特征		说 明	例 子
前缀	F	浮点数操作	FADD
	FI	整数操作	FIADD
	FB	BCD 码数操作	FBADD
	FN	不进行异常检测和处理	FNINIT
后缀	P	使堆栈弹出一次	FADDP
	PP	使堆栈弹出两次	FCOMPP

電子工業出版社

表 3-22　浮点指令集

指令类别		功　能
浮点传送类	取数	取数据存入 ST(0)
	存数	将 ST(0)保存到栈顶或存储器
	存数出栈	除存数外，还要弹出栈顶
	交换	将栈顶 ST(0)与数据寄存器 ST(*i*)之间的数据交换
	常数传送	将经常使用的常数，如 0、1、π 等按扩展精度压入栈顶 ST(0)
算术运算类	算术运算	实现浮点数、16/32 位整数的加、减、乘、除运算
	算术运算相关	对 ST(0)、ST(1)求绝对值、取整等
超越函数类	三角函数	对 ST(0)求正弦、余弦、正切；对 ST(1)/ST(0)求反正切
	指数	计算 2^x-1，x 取自栈顶 ST(0)，结果返回栈顶，x 必须介于±1.0 之间
	对数	计算 log2(st(0))×st(1))，结果送 st(1)，并出栈
浮点比较类	基本比较	比较 ST(0)与指定的源操作数，结果记录在浮点状态寄存器 C3/C2/C0 中
	设置整数状态标志	比较 ST(0)与 ST(1)，并设置整数状态寄存器 PF/ZF/CF 位
	比较传送	若条件 cc（同 SET 指令中的 cc）成立，则 ST(0)←ST(*i*)，否则不传送
FPU 控制类	系统控制	对 FPU 的操作方式进行设置：初始化、异常清除、中断控制等
	环境控制	对 FPU 的浮点寄存器组进行保存和设置

习 题 3

3-1　80x86/Pentium CPU 的有效地址 EA 由哪 4 个分量组成？它们可组合出哪些存储器寻址方式？试讨论各种存储器寻址方式与 EA 计算公式的关系。

3-2　设 BX=637DH，SI=2A9BH，位移量 D=7237H，试求下列寻址下有效地址 EA=？

（1）直接寻址　　（2）基址寻址　　（3）使用 BX 的间接寻址

（4）基址加变址寻址　　（5）带位移量的基址加变址寻址

3-3　指出下列指令中源操作数的寻址方式。若是存储器寻址，试用表达式表示出有效地址 EA=？物理地址 PA=？

（1）MOV　SI, 2100H　　（2）MOV　CX, DISP[BX]

（3）MOV　[SI], AX　　（4）ADC　AX, [BX][SI]

（5）MOV　CX, [BP]　　（6）MOV　AX, [BX+10H]

（7）MOV　AX, ES:[BX]　　（8）MOV　AX, [BX+SI+20H]

3-4　指出下列指令的源操作数字段是什么寻址方式？

（1）MOV　EAX, EBX　　（2）MOV　EAX, [ECX][EBX]

（3）MOV　EAX, [ESI][EDX*2]　　（4）MOV　EAX, [ESI*8]

3-5　指出下列传送类指令中非法的指令（针对 8086/8088 CPU）。

（1）MOV　BX, AL　　（2）MOV　BH, AL

（3）MOV　100, CL　　（4）MOV　SS, 2400H

（5）MOV　[SI], [BX]　　（6）MOV　[SI+4*3], BX

（7）MOV　AX, [BX][BP]　　（8）MOV　AX, [SI][BP]

（9）MOV　AX, BX+DI　　（10）MOV　AX, [BX+DI]

電子工業出版社

（11）MOV　SP, AX　　（12）MOV　SP, [AX]

（13）XCHG　AH, AL　　（14）XCHG 200, AL

（15）OUT　21H, AL　　（16）OUT　260H, AL

3-6　分析下列指令在语法上是否有错，如有错，请指出错误所在（针对 8086/8088 CPU）。

（1）SUB AX, [BX+BP]　　（2）ADD　VALUE1, VALUE2

（3）MOV　DS, 1000H　　（4）MOV　CS, AX

（5）CMP1000H, BX　　（6）MOV　VALUE1, CX

（7）MOV　DS, ES　　（8）ROL　[BX][DI], 3

（9）NOTBX, DX　　（10）PUSH　CS

（11）PUSH　2A00H　　（12）IN　100H, AL

（13）LEA　BX, 4300H　　（14）MOV　AX, [8-BX]

（15）MOV　AX, [BX-SI]　　（16）INC　[BX]

3-7　若 SP=2000H，AX=3355H，BX=4466H，执行下列指令或程序段后有关寄存器的内容是什么？

（1）PUSH　AX　　执行后，AX=？SP=？

（2）PUSH　AX
PUSH　BX
POP DX　} 执行后，AX=？DX=？SP=？

3-8　设 BX=6F30H，BP=0200H，SI=0046H，SS=2F00H，[2F246H]=4154H，执行指令 XCHG BX, [BP+SI]后，BX=？[2F246H]=？

3-9　设 DS=C000H，[C0010H]=0180H，[C0012H]=2000H，执行 LDS　SI, [10H] 指令后，SI=？DS=？

3-10　已知 DS=091DH，SS=1E4AH，AX=1234H，BX=0024H，CX=5678H，BP=0024H，SI=0012H，DI=0032H，[09226H]=00F6H，[09228H]=1E40H，[1E4F6H]=091DH，试求单独执行下列指令后的结果。

（1）MOV　CL, 20H[BX][SI]　　CL=？

（2）MOV　[BP][DI], CX　　[1E4F6H]=？

（3）LEA　BX, 20H[BX][SI]　　BX=？

MOV　AX, 2[BX]　　AX=？

（4）LDS　SI, [BX][DI]

MOV　[SI], BX　　SI=？　[SI]=？

（5）XCHG　CX, 32H[BX]

XCHG　20H[BX][SI], AX　　AX=？　[09226H]=？

3-11　设 SS=0915H，DS=0930H，SI=0A0H，DI=1C0H，BX=80H，BP=470H。若指令“MOV AX, OPRD”中源操作数的物理地址为 095C0H，试用 4 种不同寻址方式改写此指令，并要求使用上述条件至少一次。

3-12　已知 AX=8060H，DX=580H，端口 PORT1 的地址为 40H，内容为 4FH，端口 PORT2 的地址为 44H，指出执行下列指令后的结果在哪儿？为多少？

（1）OUTDX, AL　　（2）OUT　DX, AX

（3）IN　AL, PORT1　　（4）IN　AX, 40H

電子工業出版社

（5）OUT PORT2, AL　　（6）OUT　PORT2, AX

3-13　写出指令序列，将数据段内首地址为 BLOCK 的字数组的第 7 个字送到 CX 寄存器。

（1）使用直接寻址方式　　（2）使用 BX，寄存器间接寻址方式

（3）使用 BX，基址寻址方式　　（4）使用 BX 和 SI，基址加变址寻址方式

3-14　已知 0～9 数字的 ASCII 码表首址是当前数据段的 0A80H，现要找出数字 5 的 ASCII 值，试写出用指令 XLAT 进行翻译的指令序列。

3-15　下列每组中的两条指令有什么区别？其中，W 为数据段中的字变量，N 为不超过 16 位的常量。

（1）MOV	AX, 1000H	MOV	AX, DS:[1000H]
（2）MOV	AX, N	MOV	AX, W
（3）MOV	AX, [BX+N]	MOV	AX, [BX+W]
（4）MOV	[BP][SI], CL	MOV	DS:[BP][SI], CL
（5）MOV	AL, 5	IN	AL, 5
（6）MOV	AX, W	LEA	AX, W
（7）AND	AX, BX	TEST	AX, BX
（8）SUB	AX, BX	CMP	AX, BX
（9）INC	SI	ADD	SI, 1
（10）NEG	AX	NOT	AX

3-16　已知数据如图 3-6 所示，数据是低位在前，按下列要求编写程序段：

（1）完成 NUM1 和 NUM2 两个双精度字数据相加，和放在 NUM2 开始的双字单元中。

（2）完成 NUM2 单元开始的连续 4 字节数据相加，和为 16 位数，放在 RES 和 RES+1 两个单元中。

3-17　已知压缩型 BCD 数如图 3-6 所示，低位在前，按下列要求编写计算 BCD 数据的程序段。

（1）完成从 NUM2 单元开始的连续 4 个压缩型 BCD 数相加，和（超过 1 字节）放在 RES 和 RES+1 两单元中。

（2）完成 NUM1 单元和 NUM2 单元开始的两个 4 位 BCD 数相减（4148−2258），差存入 RES 和 RES+1 单元。

	内存
DS	
NUM1	48H
	41H
	16H
	28H
NUM2	58H
	22H
	52H
	84H
RES	

图 3-6　习题 3-16 图

3-18　已知数据如图 3-6 所示，低位在前，按下列要求编写程序段：

（1）NUM1 和 NUM2 两字节数据相乘（均为无符号数），乘积放在 RES 开始的单元。

（2）NUM1 和 NUM2 两个字数据相乘（均为带符号数），乘积放在 RES 开始的单元。

（3）NUM1 单元的字节数据除以 46（均为无符号数），商和余数依次放入 RES 开始的 2 个字节单元。

（4）NUM1 单元的字数据除以 NUM2 单元的字（均为带符号数），商和余数依次放入 RES 开始的 2 个字单元。

3-19　试用 1 条指令测试 AL 的内容并置标志位，但 AL 的内容不变，请给出多种解答。测试后，如何判断 AL 是否为 0？如何判断 AL 是正数还是负数？如何判断 AL 中有偶数个 1 还是奇数个 1？

電子工業出版社

3-20　设 AL=0B7H。写出依次执行下列每条指令后 AL 的值。

（1）SAR　AL, 1　　（2）SHL　AL, 1

（3）ROR　AL, 1　　（4）RCL　AL, 1

3-21　写出下列各程序段执行后 AL 的值。

（1）
```
MOV　　AL, 1
ROR　　AL, 1
OR　　AL, AL
NEG　　AL
```

（2）
```
MOV　　AL, 0FH
XOR　　AL, 0FFH
MOV　　CL, 7
SHR　　AL, CL
```

（3）
```
MOV　　AL, BL
NOT　　AL
XOR　　AL, BL
OR　　AL，BL
```

（4）
```
MOV　AL, 34H
STC
OR　　AL, 3
ADC　 AL, 1
```

3-22　编写程序段，用移位和相加的方法将 AL 中的无符号数乘以 21 后存入 AX。

3-23　按下列要求写出相应的指令或程序段：

（1）屏蔽 BX 寄存器的 b_4、b_6、b_{11} 位。

（2）使 CX 寄存器中的整数变为奇数。

（3）将 AX 寄存器的 b_5、b_{14} 位取反，其他位不变。

（4）测试 DX 寄存器的 b_0、b_9 位是否为 1。

（5）用 2 条指令将 AL 的 0、2、4、6 位求反，并将 1、3、5、7 位置 1。（多种解答）

（6）将 AL 中压缩 BCD 数拆成 2 个非压缩 BCD 数，并存入 AX。

3-24　若 AL=FFH，BL=03H，列表说明下列各指令单独执行后标志 OF、SF、ZF、PF、CF 的状态。

（1）ADD　BL, AL　　（2）SUB　BL, AL

（3）NEG　BL　　（4）CMP　BL, AL

（5）MUL　BL　　（6）IMUL　BL

（7）INC　AL　　（8）AND　BL, AL

（9）OR　BL, AL　　（10）XOR　BL, BL

（11）SAR　AL, 1　　（12）SHR　AL, 1

3-25　分别指出以下两个程序段的功能。

（1）
```
MOV　　CX, 10
LEA　　SI, FIRST
LEA　　DI, SECOND
REP　　MOVSB
```

（2）
```
CLD
LEA　　DI, [0404H]
MOV　　CX, 0080H
XOR　　AX, AX
REP　　STOSW
```

3-26　将数据段中长度为 20 的字串 STRING 送到附加段中首地址为 DESTIN 的缓冲区中。

3-27　已知字符串 STRING 包含 32 KB 的内容，将其中的“$”替换成空格。

3-28　设 DS=2000H，BX=1256H，SI=528FH，位移量 TABLE=20A1H，[232F7H]=3280H，[264E5H]=2450H，执行下列指令：

（1）JMP　BX　　; IP=?

（2）JMP　TABLE[BX]　　; IP=?

電子工業出版社

（3）JMP　　[BX][SI]　　; IP=?

3-29　设当前断点处的 IP=3D8FH，CS=4050H，SP=0F17CH，当执行 CALL　2000:009AH 后，试指出 IP、CS、SP、[SP]、[SP+1]、[SP+2]、[SP+3]的内容是什么？

3-30　编写程序段，根据 AL 中的内容决定程序的走向：若 b_0 是 1，其他位为 0，转向 LAB1；若 b_1 是 1，其他位为 0，则转向 LAB2；若 b_2 为 1，其他位为 0，则转向 LAB3；若 b_2～b_0 都是 0，则顺序执行。假定所有的转移都是短转移。

3-31　编程检测 50H 端口输入的字节数据，若为正，将 BL 清 0；若为负，将 BL 置为 FFH。

3-32　已知 X、Y、Z 为内存中三个带符号的字变量，编程实现：

（1）如果三个数都不相等，则将 S 单元置 0；

（2）如果三个数中有两个相等，则将 S 单元置 1；

（3）如果三个数都相等，则将 S 单元置 2。

3-33　统计以“$”结尾的字符串 STRING 的字符个数。

3-34　统计一个首地址为 BASE 的 20 字数据块中负元素个数，结果存放在 RESULT 单元。

3-35　编写实现下列功能的程序段：

（1）EDX:EAX 中的 64 位数据左移一位。

（2）比较 EAX、EBX、ECX 中无符号数的大小，并将最小的数存入 EAX 中。

3-36　用 TEST 与 SETZ 指令组合，测试 EAX 中的 8 位 16 进制数是否有一位为 0，用 BH=0 表示没有一位为 0。

3-37　按照如下要求完成浮点程序段：

（1）确定栈顶数据是否等于寄存器 ST(2)。

（2）把寄存器 ST(3)的数据加到栈顶。

（3）将栈顶数据减去寄存器 ST(4)的数据，结果保存在寄存器 ST(4)。

（4）把 π 压入栈顶。

電子工業出版社

第 4 章　汇编语言程序设计

汇编语言具有执行速度快和易于实现对硬件控制等独特的优点，至今仍然是实时控制等应用软件经常使用的程序设计语言。本章主要基于 80x86/Pentium CPU 的指令系统和 Microsoft MASM 汇编程序，介绍汇编语言程序的基本结构、语法规则和基本设计方法。

4.1　程序设计语言概述

用于计算机程序设计的语言可分为机器语言、汇编语言和高级语言三种。

每种 CPU 都有自己的指令系统，根据指令来完成各种操作。当指令和数据都用二进制编码表示时，CPU 能够直接识别与执行，因而称为机器语言。机器语言是“面向机器”的，编写、阅读和交流都非常困难，且随 CPU 的型号不同而异，通用性差。然而，无论人们使用什么语言编写程序，最终都需要转换成机器语言由 CPU 执行。

高级语言是“面向过程”的语言，采用接近人类习惯的自然语言和数学语言编写程序，不依赖具体机器。用这种语言编程，程序员可以完全不考虑计算机的硬件结构与性能，不必深入了解其指令系统，编写的程序与问题本身的数学模型之间有着良好的对应关系，且便于移植。但是，高级语言源程序并不能在计算机上直接执行，需要经编译程序将它们翻译成对应的目标程序（即机器语言程序），计算机才能运行。

通常，高级语言的一条语句相当于多条机器语言指令，编译后生成的目标程序比较大，占用内存多，执行时间长。加之高级语言源程序是在未考虑计算机硬件结构特点的条件下编写的，因此不能充分利用 CPU 的某些具体特性，限制了它在某些场合的运用，如实时数据采集、检测和控制系统等。

汇编语言是一种采用助记符表示的程序设计语言，是将机器指令符号化了的编程语言，又称符号语言。汇编语言也是面向机器的，通常，一个助记符表示一条机器指令，是英文单词的缩写（如 MOV、ADD 等），方便人们书写、阅读、记忆和检查。由汇编语言编写的源程序就是机器语言程序的符号表示，汇编语言源程序与其对应的目标程序之间有着明显的一一对应关系。

用汇编语言编写程序能够利用计算机系统的硬件特性（如寄存器、标志、中断等），直接对位、字节、字、寄存器、存储单元或 I/O 端口进行处理；能够直接使用 CPU 指令系统提供的各种寻址方式，编写出高质量的程序。这样的程序占用内存空间少、执行速度快。所以汇编语言多用在一些对内存容量和运行速度要求比较高的编程场合，如系统软件、实时通信、实时控制、I/O 接口驱动等程序的设计，或者在大型软件设计中与高级语言混合使用。当然，由于汇编语言源程序和所要解决问题的数学模型之间的关系不够直观，使得汇编语言程序设计难度增大、时间较长、出错的可能性也增大。同时，由于汇编语言与具体的计算机硬件结构有关，进行汇编语言程序设计时不仅要考虑程序本身的结构，还要考虑计算机硬件资源的使用。所有这些都对程序设计人员有较高的要求。

電子工業出版社

汇编语言源程序在交付计算机执行之前也需要先翻译成目标程序，这个编译过程称为汇编，完成汇编任务的工具被称为汇编程序（Assembler）。汇编程序是计算机系统软件之一，除了能够将汇编语言源程序翻译成机器代码这一主要功能，还能够根据用户要求自动分配存储区域（包括程序区、数据区等）；自动地将各种进制数转换成二进制数、将字符转换成ASCII 码、计算表达式的值；自动对源程序进行检查，给出错误信息（如非法格式、未定义的助记符、标号、漏掉操作数）等。它提供汇编语言源程序的所有语法规则，所以在使用汇编语言编程之前应首先熟悉相应的汇编程序。

因为汇编语言源程序在编程的便利性和可读性等方面的弱点，现在人们常常选择 C 语言编写应用程序，完整的编译汇编过程如图 4-1 所示。

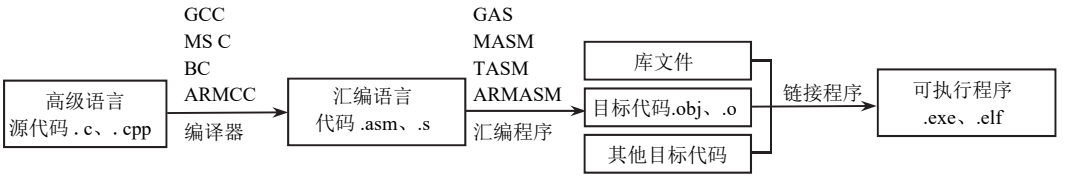


图 4-1　C 语言程序的上机与处理过程

在 C 语言源程序变成可执行目标代码过程中，需要经过编译和汇编两个阶段，常用的汇编程序有以下几种。

① GAS：GNU 汇编器，属于 GCC 编译器的一部分。GCC（GNU Compiler Collection，GNU 编译器套件）是由 GNU 组织开发的编译器，现已被大多数类 UNIX 操作系统（如 Linux、Mac OS X 等）采纳为标准的编译器。GAS 是一组汇编器，可以编译不同处理器（x86、ARM、MIPS 等）的汇编语言，最新版本为 2.31。初始 GAS 只支持 AT&T 汇编格式，从 2.10 版本起可以通过 intel_syntax 伪指令来使用 Intel 汇编格式。

② MASM：微软公司为 Intel x86 系列微处理器开发的宏汇编程序，除了支持基本汇编功能，还支持宏操作、条件汇编、结构、记录等高级宏汇编语言功能。MASM 宏汇编程序不断升级，MASM 4.0 可支持 8086/8088 到 80286 CPU 指令，5.0 可支持 80386 CPU 指令，6.11 可支持 Pentium CPU 的指令集。在 MASM 6.11 的基础上，只要增加补丁程序，即可升级到支持 MMX 指令集的 MASM 6.13、支持 SSE、SSE2 和 SSE3 指令集的 MASM 6.14 和 MASM 6.15 等。MASM 7.0 集成于 Visual C++ .NET 2002，MASM 8.0 集成于 Visual C++ .NET 2005，也可以汇编 64 位的代码。MASM 14.0 集成于 Visual Studio 2015 及其以后的版本中。

③ TASM：Borland 公司的汇编编译器，也是一种使用很广泛的编译器，性能同 MASM。与 MASM 相比，TASM 的升级没有那么频繁，早在 1.0 版本就有了对 80386 CPU 指令的完全支持，4.0 版是 TASM 系列编译器编写 MS-DOS 程序所使用的最广泛版本。TASM 的最后一个版本是 5.0，现在它已经停止更新。

④ ARMASM：ARM 公司推出的汇编器，支持所有 ARM 处理器。ARMASM 集成于 ARM 公司的 Keil 开发环境，最新的版本为 6.11，增加了对 ARM v8 指令集的支持。

由汇编程序产生的目标模块是属性为 .obj 的二进制文件，但它仅是浮动的目标程序，不能直接上机运行。必须经链接程序（LINK）把目标文件与库文件以及其他目标文件链接在一起，形成属性为 .exe 的可执行文件，并交给操作系统装入内存执行。

图 4-1 中的 .s、.o、.elf 分别是 Linux 操作系统下汇编语言代码、目标代码及可执行程序的扩展名。

電子工業出版社

4.2 汇编语言的程序结构与语句格式

一个源程序必须用其编译器能够理解的语法进行编写，这些语法是指语句和程序结构必须遵循的规则，汇编语言源程序的编写也是如此。从本节开始介绍 Microsoft MASM 汇编程序的基本语法规则。

4.2.1 汇编语言源程序的框架结构

首先，通过以下可在 MS-DOS 环境下运行的 8086/8088 汇编语言源程序实例，初步认识标准的具有完整段定义格式的汇编语言源程序的框架结构。

【例 4-1】 打印输出字符串“THIS IS A SAMPLE PROGRAM.”。

```
; SAMPLE PROGRAM  DISPLAY  MESSAGE          ; 注释行
DATA     SEGMENT                            ; 定义数据段
MESSAGE  DB  'THIS IS A SAMPLE PROGRAM. '   ; 在存储器中存放供显示的数据
         DB  0DH, 0AH, '$'
DATA     ENDS
STACK    SEGMENT  PARA  STACK  'STACK'      ; 定义堆栈段
         DB  1024  DUP (0)                  ; 在存储器的某个区域建立一个堆栈区
STACK    ENDS
CODE     SEGMENT                            ; 定义代码段
         ASSUME  CS:CODE, DS:DATA, SS:STACK    ; 段寄存器说明
MAIN     PROC     FAR                       ; 将程序定义为远过程
START:   PUSH     DS                        ┐
         MOV      AX, 0                     ├; 标准序，以便返回操作系统
         PUSH     AX                        ┘
         MOV      AX, DATA                  ; 取数据段段地址
         MOV      DS, AX                    ; 建立数据段的可寻址性
         LEA      BX, MESSAGE               ; MESSAGE 地址偏移量→BX
LOOP1:   CMP      BYTE PTR [BX], '$'
         JE       LOOP2
         MOV      AH, 5                     ┐
         MOV      DL, [BX]                  ├; 5 号 DOS 系统功能调用打印输出字符
         INT      21H                       ┘
         INC      BX
         JMP      LOOP1
LOOP2:   RET                                ; 返回 DOS 操作系统
MAIN     ENDP                               ; 过程结束
CODE     ENDS                               ; 代码段结束
         END      START                     ; 程序从 START 开始，汇编结束
```

鉴于 80x86/Pentium CPU 采用存储器分段管理方式，其汇编语言源程序都以逻辑段为基础，按段来组织代码和数据。所以 80x86/Pentium 汇编语言源程序在结构上具有以下特点。

（1）源程序由若干逻辑段组成，各逻辑段都有一个段名，由段定义语句（伪指令语句）来定义和说明。

80x86/Pentium 汇编语言源程序的这种结构是程序运行的基础，源程序一般具有代码段、

電子工業出版社

堆栈段、数据段和附加数据段。通常，只有代码段是必不可少的。数据段或附加数据段用来在内存中建立一个适当容量的工作区，以存放常数和变量，并作为算术运算或 I/O 端口传输数据的工作区等。堆栈段则在内存中建立一个堆栈区，以便在中断和子程序调用及各模块间传递参数时使用。当由这几个段构成一个完整的程序时，通常将数据段定义在代码段前面，以产生正确的机器代码。这是因为：① 程序中所使用的变量应先定义；② 汇编程序在汇编过程遇到变量时需要知道变量的属性。

某些简单的程序并不一定需要数据段和堆栈段，而对于一些复杂的程序，这几种逻辑段分别允许定义多个，但同时使用的段是有限定的。8086/8088/80286 允许同时使用 4 个：代码段（CS）、堆栈段（SS）、数据段（DS）和附加数据段（ES）；80386～Pentium 系列 CPU 允许同时使用 6 个，除了以上 4 个段，还可有 FS 和 GS 两个附加数据段。在 8086/8088 和实地址模式下，每个段的最大长度均为 64 KB；而在保护模式下，80286 允许每个段的最大长度为 16 MB，80386～Pentium 系列 CPU 允许 4 GB。

（2）在代码段的起始处，用 ASSUME 伪指令语句说明各段寄存器与逻辑段的关系，并由用户自己设置各个段寄存器（除代码段 CS 外）的初值，以建立这些逻辑段的可寻址性。

（3）每个逻辑段由若干行汇编语言语句组成，每行只有一条语句且不能超过 128 个字符，但一条语句允许有后续行，最后均以回车结束。整个源程序必须以 END 语句结束，以通知汇编程序停止汇编。END 语句后面的标号 START 表示该程序执行时的起始地址。

（4）每一条汇编语言语句最多由 4 个字段组成，均按照一定的规则分别写在一条语句的 4 个区域内，各区域之间用空格或 Tab 键隔开。

（5）每个源程序在代码段中往往需要含有返回 MS-DOS 操作系统的指令语句，以保证程序执行完毕后能自动回到 DOS 状态。终止当前程序，使其正确返回到 DOS 状态的常用方法如下。

① 标准序法。采用标准源程序框架结构编制程序：先将源程序中的主程序定义为 FAR 过程，其最后一条指令为 RET；再在主程序的开始处使用以下三条指令，则程序执行完毕后自动返回 DOS。上例中使用的就是这种“标准序”法。

```
PUSH    DS              ; 保护程序段前缀 PSP 的段地址
MOV     AX, 0           ; 保护 PSP 的 0 偏移地址
PUSH    AX
```

② 使用 DOS 的 4CH 号功能调用。不需定义主程序为 FAR 过程并去掉上述三条指令，只在代码段结束（语句 CODE　ENDS）前增加调用语句：

```
MOV     AH, 4CH         ; 功能号 4CH→AH
INT     21H             ; 中断调用
```

则程序执行完毕后自动返回 DOS，这是因为执行了功能号为 4CH 的 DOS 系统功能调用。关于 DOS 系统功能调用在后面会详细介绍。

注意：MASM 汇编程序从 5.0 版开始还支持简化段定义，通过伪指令 EXIT 返回 DOS。

4.2.2 汇编语言的语句

语句是汇编语言源程序的基本组成单位，源程序是一个语句序列。语句序列中的每条语句规定了一个基本操作要求，语句序列则完成某个特定的操作任务。

電子工業出版社

1．语句的种类与格式

（1）语句的种类

80x86/Pentium 汇编语言有三种基本语句：指令语句、伪指令语句和宏指令语句。

① 指令语句。指令语句对应 CPU 指令系统中的一条指令，是可执行语句，出现在程序的代码段中。汇编时汇编程序为之产生一一对应的机器代码，程序执行时使 CPU 产生相应功能。例如：

```
MOV     DS，AX   ；机器代码为 8EH 和 D8H
```

② 伪指令语句。伪指令语句是由伪指令构成的说明语句，可根据需要出现在任一段内，汇编时指示汇编程序如何汇编源程序，如分配存储单元、将程序分段等。伪指令语句本身并不产生机器代码，是 CPU 不执行的语句。例如：

```
SEGMENT/ENDS      ；将程序分段的信息提供给汇编程序，以不同的名字来说明是堆栈段、数
                  ；据段还是代码段
MESSAGE  DB 'THIS IS A SAMPLE PROGRAM. '
                  ；定义变量 MESSAGE 在数据段 DATA 中的存放形式。汇编时，汇编程序将
                  ；MESSAGE 定义为一个字节型数据区的首地址，并按字节存储字符串信息
```

③ 宏指令语句。对一段指令序列进行宏定义，则形成宏指令语句。汇编时，凡程序中有宏指令语句的地方都用相应指令序列的机器代码插入。宏指令语句是一般性指令语句的扩展。

（2）语句的格式

指令语句与伪指令语句的格式类似，由 4 部分组成（宏指令语句格式在稍后专门介绍）。

指令语句的一般格式：[标号:]　[前缀]　指令助记符　[操作数]　[; 注释]

伪指令语句的格式：　[名字]　伪指令定义符　[操作数]　[; 注释]

其中，“[ ]”中的内容为可选部分，操作数部分或是 0、1 个操作数，或是由“,”隔开的多个操作数。

① 标号和名字。标号和名字分别是为指令单元和伪指令所起的符号名称，是用户自定义的标识符。指令语句中的标号后面必须有“:”，伪指令语句中的名字后面不能有“:”。

标号代表指令所在存储单元的符号地址。在程序中，标号可以作为转移（JMP）、循环（LOOP）等指令的转移目标，与具体的指令地址相联系。名字一般用于定义常量、变量、过程、段名等，指示所定义变量、过程以及段的起始地址。

标号和名字的选择有一些限制。首先，它们必须符合汇编语言的标识符定义，即以字母开头，由字母（A～Z 或 a～z，汇编程序一般不区分大小写）、数字（0～9）及部分特殊字符（?、@、$和_等）组成的字符串，字符串长度不能超过 31。此外，标识符不能是汇编语言中有特定意义的保留字，如 CPU 的内部寄存器名 AL、AX 和指令助记符等。

② 助记符和定义符。助记符和定义符分别规定指令语句的操作性质和伪指令语句的伪操作功能，是语句中唯一不可省略的。指令助记符的前面可根据需要加前缀，80x86/Pentium 指令系统中允许与助记符同时使用的前缀是重复前缀 REP、REPE 及段超越前缀等。

③ 操作数。指令语句中的操作数提供该指令的操作对象，并说明要处理的数据存放在什么位置以及如何访问它。伪指令语句中操作数的格式和含义随伪操作命令的不同而不同。

④ 注释。注释由“;”开始，对程序的功能加以说明，以构成源程序的编程文档，不被汇编。

電子工業出版社

2．语句中的操作数

根据寻址方式等因素的不同，语句中操作数的表现形式有常量、寄存器、存储器、表达式。

（1）常量操作数

在汇编时，已经确定其值且程序运行期间不变化的量为常量，如语句中的立即数或端口地址等。这些常量的形式各不相同，常用的有二、八、十或十六进制的整型数值常量、字符串常量和已赋值的常量标识符（符号常量）等。

4 种整型数值常量只需在数的后面加上字母 B、D、Q 和 H 即可定义。例如，十进制数 9 可表示为以下 4 种形式：1001B、9D、9H、11Q。十进制数后面的 D 可以省略，十六进制数的第一个数字必须在 0～9 之间。因此，9D 可以写作 9；十六进制数 ABH 必须写作 0ABH。在汇编语言源程序中，常用十六进制表示数据和地址。

字符串常量是用单引号括起来的一个或多个字符，其值为每个字符对应的 ASCII 值。如'2'的值为 32H，'AB'的值为 4142H。因此，字符串常量与整型数值常量可以交替使用。

符号常量是指在程序中用标识符形式表示的常量，它的使用可提高程序的通用性。

其余常量操作数还有十进制浮点数和十六进制实数。

（2）寄存器操作数

指操作数部分是寄存器名，如 AX、SI、DS 等。

（3）存储器操作数

存储器操作数是地址型操作数，有标号和变量两种。标号是指令所存放单元的符号地址，一定在代码段内。变量一般在数据段或堆栈段中，是可以用存储器寻址方式访问的操作数。在程序中通过变量名（变量存储单元的符号地址）来引用。所以，标号和变量一经定义便具有以下三种属性。

① 段属性（SEGMENT）：标号和变量对应存储单元的段地址，是段基地址的高 16 位。

② 偏移量属性（OFFSET）：标号和变量对应存储单元的偏移地址。

③ 类型属性（TYPE）：标号的类型是指其距离属性，有 NEAR（近距离属性，用于段内转移或调用）和 FAR（远距离属性，用于段间转移或调用）两种。变量的类型是指变量占用存储单元的字节数，有 BYTE（字节属性）、WORD（字属性）和 DWORD（双字属性）等。

（4）表达式操作数

表达式由各种操作数、运算符和操作符组成，根据其表示内容的不同又分为数值表达式与地址表达式两类。汇编时，汇编程序对表达式进行运算，得到一个确定的值。

① 数值表达式和地址表达式

数值表达式由数值常量、字符串常量或符号常量等与算术、逻辑或关系运算符等连接而成，其运算结果还是一个数值，仅有大小而无其他属性，可作为指令中的立即操作数或数据区中的初值使用。

地址表达式由常量、变量、标号、寄存器（如 BX、BP、SI、DI）的内容以及一些运算符组成。其运算结果表示存储器地址，一般都是段内的偏移地址，因此它也具有段属性、偏移量属性和类型属性。如 ES:[SI+4]、PLACE+2*3 这样的地址表达式操作数。

② 运算符和操作符

MASM 宏汇编中有三类运算符（算术、逻辑和关系运算符）和两类操作符（分析和合成操作符），如表 4-1 所示。运算符实现对操作数的相关运算，操作符则完成对操作数属性的定义、调用和修改。

電子工業出版社

算术运算符实现加、减、乘、除和求余的算术运算。它们都可用于数值表达式，只有+、-运算符可用于地址表达式。用于地址表达式的形式是“标号或变量±常量”，当标号或变量加上或减去某个常量时，结果仍为标号或变量，其三种属性中的类型与段地址属性不变，偏移量属性被修改。例如，PLACE+2*3 是指 PLACE 字节单元后的第 6 个存储单元的地址。

逻辑运算符实现按位相与、相或、异或、求反的逻辑运算，只适用于数值表达式。

逻辑运算符和逻辑运算指令助记符在符号形式上是一样的，但两者的含义有本质差异。作为逻辑运算符时，它们是在程序汇编时由汇编程序计算的，计算结果构成指令操作数的一部分，且逻辑运算符的操作对象只能是整型常量；而作为指令助记符时，是在程序运行中被执行的，操作对象还可以是寄存器或存储器操作数。

表 4-1　MASM 支持的运算符和操作符

类　型		符　号	名　称	运算结果	实　例
运算符	算术	+	加法	和	3+5=8
		-	减法	差	8-3=5
		*	乘法	乘积	3*5=15
		/	除法	商	22/5=4
		MOD	模除	余数	12 MOD 3=0
		SHL	左移	左移后的二进制数	0010B SHL 2=1000B
		SHR	右移	右移后的二进制数	1100B SHR 1=0110B
	逻辑	NOT	非运算	逻辑非结果	NOT 1010B=0101B
		AND	与运算	逻辑与结果	1011B AND 1100B=1000B
		OR	或运算	逻辑或结果	1011B OR 1100B=1111B
		XOR	异或运算	逻辑异或结果	1010B XOR 1100B=0110B
	关系	EQ	相等		5 EQ 11B=全 0
		NE	不等	结果为真	5 NE 11B=全 1
		LT	小于	——输出全 1	5 LT 3=全 0
		LE	不大于	结果为假	5 LE 101B=全 1
		GT	大于	——输出全 0	5 GT 100B=全 1
		GE	不小于		5 GE 111B=全 0
操作符	分析	SEG	返回段地址	段地址	SEG N1=N1 所在段段地址
		OFFSET	返回偏移地址	偏移地址	OFFSET N1=N1 的偏移地址
		LENGTH	返回变量单元数	单元数	LEHGTH N2=N2 单元数
		TYPE	返回元素字节数	字节数	TYPE N2=N2 中元素字节数
		SIZE	返回变量总字节数	总字节数	SIZE N2=N2 总字节数
	合成	PTR	修改类型属性	修改后类型	BYTE PTR [BX]
		THIS	指定类型/距离属性	指定后类型	ALPHA EQU THIS BYTE
		段寄存器名	段前缀	修改段	ES：[BX]
		HIGH	分离高字节	高字节	HIGH 3355H=33H
		LOW	分离低字节	低字节	LOW 3355H=55H
		SHORT	短转移说明	转移距离在-128~+127	JMP SHORT LABEL
其他运算符		( )	圆括号	改变运算符优先级	(8-3)*3=15
		[ ]	方括号	下标或间接寻址	MOV　AX, [BX]
		.	点运算符	连接结构与成员变量	TAB.T1
		< >	尖括号	修改变量	<，8，5>
		MASK	返回字段屏蔽码	字段屏蔽码	MASK C
		WIDTH	返回记录宽度	记录/字段位数	WIDTH W

关系运算符用于比较和测试符号数值。MASM 用 FFH 或 FFFFH 表示关系成立（为真），用 00H 或 0000H 表示关系不成立（为假）。结果值在汇编时获得。例如：

```
MOV   BX, PORT  LT  5      ; 若 PORT 值小于 5，汇编结果为：MOV  BX, 0FFFFH
                           ; 否则，汇编结果为：MOV  BX, 0000H。
```


电子工业出版社

分析操作符与合成操作符。分析操作符又称为数值返回运算符，它的运算对象是存储器操作数，返回标号或变量的属性值。合成操作符作用于存储器操作数时可以改变它们的属性，又称为修改属性操作符。这两种操作符在后面会有详细介绍。

③ 运算符和操作符的优先权等级

运算符和操作符的优先级如表 4-2 所示。优先级相同时顺序为先左后右。

表 4-2　运算符和操作符的优先级

优 先 级		运算符和操作符
高	1	LENGTH，SIZE，WIDTH，MASH，( )，[ ]，<>，．
↓	2	PTR，OFFSET，SEG，TYPE，THIS，段寄存器名：（加段前缀）
	3	HIGH，LOW（操作数高、低字节）
	4	+，－（单目）
	5	*，/，MOD（求模），SHL，SHR
	6	+，－（双目）
	7	EQ，NE，LT，LE，GT，GE
	8	NOT
	9	AND
	10	OR，XOR
低	11	SHORT

4.3　汇编语言的伪指令

MASM 宏汇编提供以下功能的伪指令：选择处理器方式、存储模式定义、符号定义、数据定义、程序分段、过程和宏定义、模块定义、条件汇编、结构定义以及列表控制等。除了数据定义伪指令，其余伪指令都不分配存储空间，仅起说明作用。本节分别介绍基本伪指令（支持 8086/8088）和扩展伪指令（MASM 5.0 及以上版本，支持 80386/Pentium）的功能和使用方法。

4.3.1　基本伪指令语句

1．符号定义伪指令

符号定义伪指令用于给程序中的常量或表达式赋予一个符号名，方便它们在程序中以此符号名的形式出现，提高程序的可读性与灵活性。符号定义伪指令有 EQU 和 = 两种。

（1）等值语句 EQU

格式：符号名　EQU　表达式

例如：

```
PORT1   EQU   312          ; 给常量赋予符号名
PORT2   EQU   PORT1+1      ; 给数值表达式赋予符号名 PORT2
ADDR    EQU   ES:[SI+4]    ; 给地址表达式定义一个名字
CHAR    EQU   'COMPUTER'   ; 为字符串定义新的名字
COUNT   EQU   CX           ; 为寄存器 CX 定义新的符号名 COUNT
LD      EQU   MOV          ; 为 MOV 定义新的符号名 LD
```

注意：在同一源程序中，同一符号不能用 EQU 重复定义。

電子工業出版社

（2）等号语句 =

格式：符号名 = 表达式

此语句的功能与 EQU 类似，唯一区别是能对同一符号重复定义。

例如：

```
EMP=60                          ; 定义 EMP 等于 60
EMP=EMP+1                       ; 重新定义 EMP 等于 61
EMP=123*4                       ; 重新定义 EMP 等于 492
```

2. 数据定义伪指令

数据定义伪指令用于定义变量，指定变量的类型和名称，为变量分配存储单元并赋初值。

（1）数据定义伪指令语句的格式

格式：[变量名]　数据定义符　操作数 [, 操作数]…[, 操作数]

其中，变量名是可选的；操作数是赋给变量的初值，多个相同类型的变量可在一条语句中定义；常用的“数据定义符”是下列三种伪指令之一。

① DB：定义字节变量，每个变量分配 1 字节的存储单元。

② DW：定义字变量，每个变量分配 2 字节存储单元，且遵循“数据的低位部分在低地址，高位部分在高地址”的存放规则。

③ DD：定义双字变量，每个变量分配 4 字节存储单元，存放时同样按“低位字在低地址，高位字在高地址”的存放规则。

其他还有 DQ 和 DT 两种整型数据定义符，DQ 定义 4 字变量，DT 定义 10 字节变量。MASM 6.x 还支持实数定义，如数据定义符 REAL4、REAL8 分别用于定义 4、8 字节的实型变量。

经过定义的变量名有三种属性：段地址（可用操作符 SEG 获得）、偏移量（可用操作符 OFFSET 获得）和数据类型（字节、字或双字等）。若某个变量所表示的是一个数组，则其数据类型属性为变量的单个元素所分配的字节数。

（2）数据定义伪指令语句的具体形式和功能

① 为数据项分配存储单元，变量名作为该存储单元的名称。例如：

```
X       DB      25H             ; 定义 X 为字节变量
Y       DW      4142H           ; 定义 Y 为字变量
Z       DD      12345678H       ; 定义 Z 为双字变量
```

汇编时汇编程序将 25H、4142H、12345678H 这些初值分别放入名为 X、Y、Z 的存储单元中，如图 4-2 所示。

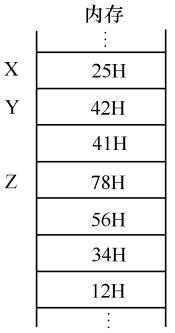


图 4-2　汇编后存储器的分配情况

初始值也可以是一个数值表达式，因为表达式的值是在汇编时计算的。例如：

```
IN_PORT    DB      PORT_VAL             ; PORT_VAL 是符号常量，已由之前的 EQU 语句赋值
OUT_PORT   DB      PORT_VAL+1
```

② 预留若干字节（或字、双字等）存储单元但并不赋具体的初值，用于存放程序的运行结果。这时操作数部分可使用“?”。例如：

```
A       DB      ?               ; 为变量 A 分配 1 字节单元
B       DW      ?               ; 为变量 B 分配 1 个字单元
```

③ 引入若干用“,”分隔的操作数来定义一个表（数组）。

例如：

電子工業出版社

```
W_TABLE      DW    1122H, 3344H            ; 定义一个字表
B_TABLE      DB    1, 2, 4, 8, 16          ; 定义一个字节表
```

其中，字变量 W_TABLE 的内容为 1122H，下一个字单元（起始地址 W_TABLE+2）的内容为 3344H。字节变量 B_TABLE 的内容为 1，B_TABLE+1 单元的内容为 2，其余以此类推。

当表中的操作数相同时，可以用重复操作符 DUP 来缩写。

例如：ALL_ZERO　DB　0，0，0，0，0，0 可以写成　ALL_ZERO　DB　6 DUP(0)。

DUP 是重复数据定义操作符，利用给出的一个初值（或一组初值）以及这些值应该重复的次数（由 DUP 前面的常数决定）来初始化存储器。例如：

```
W_TABLE      DW    10H DUP(?), 20H DUP(7)  ; 定义 30H 个字单元
B_TABLE      DB    10H DUP(12H, 34H)       ; 重复存放 10H 字节 12H、34H，共占用 20H 字节单元
```

DUP 操作符还可以嵌套使用。例如：

```
; 定义了 0、1、2、1、2、1、2、3 的 2 个副本，共占用 16 字节单元
ARRAY        DB    2   DUP(0, 3 DUP(1, 2), 3)
```

④ 用 DB 伪指令在内存中定义一个字符串。字符串中的每个字符用它的 ASCII 值来表示，为 1 字节，所以字符串的定义必须用 DB 命令。例如：

```
MESSAGE      DB    'HELLO'                 ; MESSAGE 所指的单元中存放字符 H 的 ASCII 值
                                           ; MESSAGE+1 单元中存放字符 E 的 ASCII 值……
                                           ; MESSAGE+4 单元中存放字符 O 的 ASCII 值
```

当字符串的长度不超过两个字符时，也可用 DW 来定义。例如：

```
STRING1      DB     'AB'
STRING2      DW     'AB'
STRING3      DW     'AB', 'CD'
```

这三条语句在汇编后，存储器初始化的情况如图 4-3 所示。

⑤ 当操作数是标号或变量时，可用 DW 或 DD 伪指令将标号或变量的偏移地址或全地址来初始化存储器。例如：

```
XX           DW    CYCLE                   ; 变量 XX 的初值为 CYCLE 的偏移地址
YY           DD    CYCLE                   ; 变量 YY 的初值为标号 CYCLE 的段地址和偏移地址
        …
CYCLE:       MOV   BX, AX                  ; CYCLE 是程序中一条指令语句的标号
```

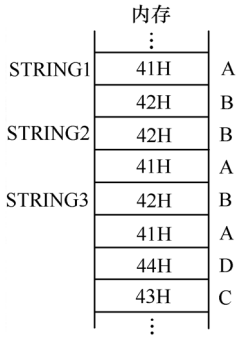


图 4-3　字符串定义的存储形式

第 1 条伪指令将标号 CYCLE 的 16 位偏移地址存放在与变量 XX 对应的 2 字节存储单元内；第 2 条伪指令在与变量 YY 对应的 4 字节存储单元内先存放 CYCLE 的 16 位偏移地址，再存放 CYCLE 的 16 位段地址。

⑥ 数据定义语句定义了变量的类型，能使汇编程序对访问存储器的指令产生正确的目标代码。例如：

```
SUM  DW    ?                             ; 定义 SUM 为字变量
           …
           INC     SUM                   ; 将变量 SUM 的字内容加 1
```

前已述及，当“标号或变量±常量”时，形成的新操作数与原操作数有相同的类型属性。如对上述举例有：SUM+2 是字型变量，CYCLE+1 也是一条指令语句的标号。

（3）分析操作符与合成操作符

作为存储器操作数的标号或变量都有段、偏移量和类型三种属性。在程序设计中，有

電子工業出版社

时需要随时查询、调用或修改这些属性，以满足不同的访问要求。这些操作由分析与合成操作符来完成。

① 分析操作符：作用于存储器操作数时返回它们的属性值，这些操作符是 SEG、OFFSET、TYPE、SIZE、LENGTH 等。

使用格式：　操作符　标号或变量

SEG 和 OFFSET 操作符：分别用来返回标号或变量的段地址和偏移地址。例如：

```
ARRAY    DB      100  DUP(0)              ; 定义字节变量数组
         …
         MOV     AX, SEG  ARRAY          ; 取变量 ARRAY 的段地址→AX
         MOV     DS, AX                  ; AX→DS
         MOV     BX, OFFSET  ARRAY       ; 取 ARRAY 的偏移地址→BX
         MOV     AL, [BX]
```

设数据段 DATA 的起始地址为 06000H，字节变量 ARRAY 在 DATA 段内，则上述指令被汇编时，SEG ARRAY 回送 ARRAY 所在段的段地址 0600H，OFFSET ARRAY 回送 ARRAY 在段内的偏移量 0000H。CPU 执行指令时，立即数 0600H 被送入 AX，0000H 被送入 BX。

TYPE 操作符返回一个数值，表示存储器操作数的类型。各种存储器操作数的类型值如表 4-3 所示。LENGTH 操作符返回变量用 DUP 重复定义的数据项总数，其他情况返回 1。SIZE 操作符返回 TYPE 和 LENGTH 的乘积，表示为变量所分配的字节单元总数。

表 4-3　存储器操作数的类型值

存储器操作数	类型属性	类 型 值	说　明
字节变量	BYTE	1	变量类型的每个数据占用的字节数
字变量	WORD	2	
双字变量	DWORD	4	
标号	NEAR	−1	距离属性值（没有实际物理意义）
标号	FAR	−2	

例如：

```
MOV  AX, TYPE  ARRAY      ; ARRAY 是字节变量，该语句汇编结果为 MOV  AX, 1
```

例如：

数据定义	LENGTH 操作符	SIZE 操作符
BUFFER1　DB　01H, 02H	LENGTH　BUFFER1=1	SIZE　BUFFER1=1×1=1
BUFFER2　DW　1234H	LENGTH　BUFFER2=1	SIZE　BUFFER2=2×1=2
BUFFER3　DD　100 DUP(20H)	LENGTH　BUFFER3=100	SIZE　BUFFER3=4×100=400
BUFFER4　DB　'ABCD……Z'	LENGTH　BUFFER4=1	SIZE　BUFFER4=1×1=1

② 合成操作符：又称为修改属性操作符，作用于存储器操作数时，能对其类型进行设置，建立一些新的存储器操作数以满足不同的访问要求，常用的有 PTR、LABEL、THIS 等。

PTR 操作符，用来临时指定或修改存储器操作数的类型属性，而保持其原有的段属性和偏移地址属性。如果这些变量或标号已有定义，则原定义的类型属性不变。

使用格式：类型　PTR　标号或变量

其中，类型可以是 BYTE、WORD、DWORD、NEAR、FAR 等，它们仅在当前所在的指令中有效。例如：

```
ARRAY1      DB  0, 1, 2, 3, 4              ; 定义字节变量数组
ARRAY2      DW  0, 1, 2, 3, 4              ; 定义字变量数组
```


電子工業出版社

```
        MOV   BX, WORD PTR  ARRAY1[3]          ; 使 ARRAY1 为字变量，0403H→BX
        MOV   CL, BYTE PTR  ARRAY2[6]          ; 使 ARRAY2 为字节变量，03H→CL
        MOV   WORD PTR [SI], 4                 ; 将 0004H 放入 SI 开始的字单元中
```

配合 EQU 伪指令，还可以为变量 ARRAY1 和 ARRAY2 定义新的变量名。例如：

```
WBYTE   EQU  WORD  PTR  ARRAY1
BWORD   EQU  BYTE  PTR  ARRAY2
```

新变量 WBYTE、BWORD 的段地址和偏移地址分别与 ARRAY1、ARRAY2 的相同，只是类型已改变，在之后的程序中可以直接使用新变量。例如：

```
        MOV   BX, WBYTE[3]                     ; 0403H→BX
        MOV   CL, BWORD[6]                     ; 03H→CL
```

PTR 用来指明标号或变量的类型属性时，可确定指令是段内转移还是段间转移。例如：

```
W_VAR   DD    INCHES
        …
INCHES: CMP   SUM, 100
        …
        JMP   WORD  PTR  W_VAR                 ; 段内转移
或      JMP   DWORD  PTR  W_VAR                ; 段间转移
或      JMP   NEAR  PTR  INCHES                ; 段内转移（NEAR PTR 可以略去）
或      JMP   FAR  PTR  INCHES                 ; 段间转移
```

LABEL 操作符为当前存储单元定义一个指定类型的标号或变量，使它们具有多重名字和属性。

使用格式：标号或变量名　LABEL　类型

例如：

```
WBYTE   LABEL   WORD                           ; 为变量 WBYTE 定义一个字类型的数据区
ARRAY   DB      1, 2, 3, 4                     ; 为变量 ARRAY 定义一个字节类型的数据区
        MOV     AL, ARRAY                      ; 01H→AL
        MOV     AX, WBYTE                      ; 0201H→AX
```

WBYTE 和 ARRAY 两个变量指向同一数据区，具有同样的段属性和偏移量属性，但是类型不同。这样可根据需要按不同类型去操作数据区中的数据。

THIS 操作符与 EQU 配合使用，具有 LABEL 操作符同样的作用。

使用格式：标号或变量名　EQU　THIS　类型

例如：

```
BWORD   EQU     THIS  BYTE
ARRAY   DW      100   DUP(?)
```

此例表明，可以将 100 个字的存储区 ARRAY 按 200 字节的存储区 BWORD 来使用。

3．程序分段伪指令

程序分段伪指令用于指示汇编程序和链接程序如何按逻辑段来组织程序和利用存储器，规定了源程序中段的起始和结束，并指定属性，如段的起始地址边界、同名段的组合方式等。

段定义有两类：完整段定义和简化段定义。本节仅介绍完整段定义伪指令，主要包括段定义、指定段寄存器等语句，如 SEGMENT/ENDS、ASSUME、ORG 等。

（1）段定义伪指令 SEGMENT/ENDS

SEGMENT/ENDS 这对段定义语句将程序中的指令或语句分成若干逻辑段，按性质一般分为数据段、代码段、堆栈段和附加段。

電子工業出版社

段定义格式：段名　SEGMENT　[定位类型] [组合类型] ['类别名']

…

（段体）

…

段名　ENDS

功能：指出段名及段的各种属性，并表示段的开始和结束位置。

段名是用户定义的段的标识符，是识别段的标志，表示汇编程序为该段分配的存储区的起始位置。段体是段内的指令和伪指令语句序列。三个可选参数项代表段的三个属性，为源程序的汇编与链接提供必要的信息，顺序不可交换，其意义如下。

① 定位类型。汇编程序对源程序汇编后生成的是地址浮动的目标代码文件 .obj，其中的段地址和偏移量都未最后确定，还需要用链接程序（LINK）把各模块链接起来。定位类型参数通知 LINK 程序，链接时本段起始地址的边界定位方式。定位类型有 4 种，如表 4-4 所示，默认定位类型是 PARA。

表 4-4　各种定位类型所表示的段起始地址

定位类型	含　义
PARA（节，默认状态）	段的起始地址必须为 16 的倍数（xxxx0H）
PAGE（页）	段的起始地址必须为 256 的倍数（xxx00H），就是页的起点上
WORD（字）	段的起始地址必须为偶数，最适合安排类型为字的数据段
BYTE（字节）	段可以从任意地址开始

② 组合类型。大型的汇编语言源程序可以由许多模块组成，每个模块常常有自己的数据段、代码段和堆栈段。组合类型用于通知 LINK 程序，在对多个模块进行链接形成一个可执行文件时，如何将本段与其他模块中的同名段进行合理组合。组合类型有 6 种，如表 4-5 所示。

表 4-5　各种组合类型表示的链接关系

组合类型	含　义
NONE（默认状态）	本段与其他模块中的同名段无逻辑关系，不组合，各自有自己的段起始地址
PUBLIC	在满足定位类型的前提下，LINK 程序将本段与其他模块中说明为 PUBLIC 的同名段邻接在一起，共用一个段地址，即合成一个物理段
STACK	默认堆栈段需要指定的组合类型。LINK 程序将所有堆栈段链接成一个连续段（其长度是各堆栈段的长度之和），链接后自动初始化 SS、SP。如果在定义堆栈段时没有将其说明为 STACK 组合类型，则需要在程序中用指令设置 SS 和 SP 的值，此时 LINK 程序会给出一个警告信息
COMMON	各模块中由 COMMON 方式说明的同名段重叠覆盖，有着相同的起始地址。段的长度取决于最长 COMMON 段的长度，段的内容为所链接的最后一个模块中 COMMON 段的内容以及没有覆盖到的前面 COMMON 段的内容
MEMORY	本段定位在所有链接在一起的其他段的最后（存储器高地址区域）。若有多个 MEMORY 段，汇编程序则认为所遇到的第一个为 MEMORY，其余为 COMMON
AT 表达式	本段定位在表达式的值所指定的段地址上。如 AT 1930H 表示本段的起始地址为 19300H

③ 类别名：用单引号括起来的字符串，以表示该段的类别。链接时类别名相同的段在内存中相邻存放，但各段仍然是独立的。类别名可任意选择，典型的可选 STACK、CODE 和 DATA 等。若省略“类别名”，则表明该段类别为空。

（2）指定段寄存器伪指令 ASSUME

ASSUME 一般出现在代码段中 SEGMENT 伪指令的后面，用于建立段与段寄存器之间

电子工业出版社

的联系，通知 MASM 用指定的段寄存器去寻址对应的逻辑段。但是，ASSUME 语句并未真正将段地址装入相应的段寄存器，段寄存器（CS 除外）的初值设定还需在程序中用指令来完成。

格式：ASSUME　段寄存器:段名[, 段寄存器:段名, …]

下面一段程序表示如何使用 SEGMENT、ENDS、ASSUME 伪指令。

```
DATA    SEGMENT                              ; 定义 DATA 段
X       DB      ?
DATA    ENDS
ESTRA   SEGMENT                              ; 定义 ESTRA 段
ALPHA   DB      ?
ESTRA   ENDS
STACK   SEGMENT PARA STACK 'STACK'           ; 定义 STACK 段
STAPN   DB      100   DUP(?)                 ; 100 字节栈空间
TOP     EQU     LENGTH   STAPN
STACK   ENDS
CODE    SEGMENT                              ; 定义 CODE 段
ASSUME  CS:CODE, ES:ESTRA, DS:DATA, SS:STACK; 定义了 CODE、ESTRA、DATA、STACK
                                             ; 分别为代码段、附加段、数据段和堆栈段
MAIN    PROC    FAR
        PUSH    DS                           }
        SUB     AX, AX                       }; 标准序
        PUSH    AX                           }
        MOV     AX, DATA
        MOV     DS, AX                       }
        MOV     AX, ESTRA                    }
        MOV     ES, AX                       }; 给段寄存器（CS 除外）赋值
        MOV     AX, STACK                    }
        MOV     SS, AX                       }
        MOV     AX, TOP
        MOV     SP, AX                       ; 给 SP 赋值
        MOV     BL, X                        ; 把字节 X 单元的内容→字节 ALPHA 单元
        MOV     ALPHA, BL
        …
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

说明：

① 因为堆栈段定义时选用了组合类型 STACK，系统链接时就会自动初始化 SS、SP，这时应该去掉 ASSUME 语句中的 SS：STACK 以及代码段中初始化 SS、SP 的 4 条语句。

② CS 和 IP 寄存器的初值都是由伪指令语句 END　MAIN 装入的，无须在程序中设置。

END　[标号]语句是程序结束伪指令，通知 MASM，主程序或模块程序在哪儿结束，其中的标号是可选项。只有主程序的结束语句允许带标号，它是程序运行时第一条要执行的指令语句的标号。如上例中的 MAIN，它的段地址和偏移地址就是 CS 和 IP 的内容。

基于①和②，通常在程序中仅需对 DS、ES 段寄存器作初始化赋值。

電子工業出版社

③ 程序中把字节 X 的内容传送至字节 ALPHA 时使用的两条指令，因为 ASSUME 伪指令的作用，MASM 把它们汇编成以下两条等价指令：

```
MOV   BL, DS:X
MOV   ES:ALPHA, BL
```

有时为了更明确地指明段寄存器或替代 ASSUME 语句的作用，可在指令中直接增加段超越前缀，用以通知 MASM，该指令执行时应使用哪一个段寄存器。

④ 可以用 ASSUME　段寄存器:NOTHING 语句取消先前建立的段与段寄存器的联系，需要时可再重新建立段寄存器与新的段之间的关联。

（3）组定义伪指令 GROUP

伪指令 GROUP 用于将程序中若干不同名的段集合成一个组，并赋予一个组名，使它们都装在一个 64 KB 的物理段中。

格式：组名　GROUP　段名[, 段名…]

组名是识别组的标志，表示组的起始地址。段名可以是用 SEGMENT 语句定义的或者由 SEG 操作符得到的段名。如果组名在 ASSUME 语句中已说明，且相应段寄存器（DS、ES 等）有初始化语句，则系统把组内各段中的标号或变量的偏移地址调整为相对于组的起始地址。可用“OFFSET　组名:变量”得到变量相对于组名地址的偏移量，组名在这里相当于段前缀。例如：

```
DGROUP   GROUP    DATA1, DATA3              ; 定义段 DATA1、DATA3 为一个组
DATA1    SEGMENT  WORD  PUBLIC 'DATA'
X        DW       ?
DATA1    ENDS
DATA3    SEGMENT  WORD  PUBLIC 'DATA'
Z        DW       ?
DATA3    ENDS
CODE     SEGMENT  WORD  PUBLIC 'CODE'
ASSUME   DS:DGROUP, CS:CODE
START:   MOV      AX, DGROUP                ; DGROUP 为已定义的组名
         MOV      DS, AX                    ; 给 DS 赋初值
         MOV      X, OFFSET DGROUP:Z        ; 变量 Z 相对于 DGROUP 的偏移量→BX
         …
CODE     ENDS
         END  START
```

（4）指定地址伪指令 ORG

ORG 用来强行指定地址计数器（记载正在汇编的指令目标代码或数据存放在当前段内的偏移量的部件）的当前值，以改变段内在它之后的代码或数据块存放的偏移地址。

格式 1：ORG　表达式

格式 2：ORG　$+表达式　　; 符号$表示地址计数器的当前值

格式 1 直接将表达式的值（0～65535）送入地址计数器，格式 2 将地址计数器的当前值$加上表达式的值后送入地址计数器。然后依“表达式”或“$+表达式”的值为起点，来存放在 ORG 语句后的程序代码或数据，除非遇到一个新的 ORG 语句。例如：

```
DATA     SEGMENT
         ORG      10H                      ; 在数据段 10H 偏移地址处开始存放 20H、30H
X        DB       20H, 30H
         ORG      $+5                      ; 使偏移地址+5
Y        DB       40H, 50H                 ; 在数据段 17H 偏移地址处开始存放 40H、50H
```


電子工業出版社

又如：

```
ARRAY    DB    12, 34, 56
LEN      EQU   $-ARRAY                    ; LEN 的值就是 ARRAY 变量所占的字节数
```

4．过程和宏定义伪指令

过程（又称为子程序）和宏都是程序段的一种形式，汇编语言中常用定义过程或宏的方法来简化源程序，使程序结构简洁清晰，并便于建立和使用相应的库以减少编程工作量等。

（1）过程定义伪指令

过程是一段可由 CALL 指令调用的程序，执行完后用 RET 指令从过程返回到原调用处。若整个源程序由主程序和若干子程序组成，则主程序和这些子程序都应包含在代码段内，且都可以用过程定义语句来定义。

过程定义格式：

```
过程名  PROC   [NEAR]/FAR
        …
        [RET]
        …                              ; 过程体
        RET
过程名  ENDP
```

功能：定义一个过程，并指出过程名及过程的属性。

伪指令 PROC/ENDP 标示过程的起始与结束。过程名是给所定义的过程取的名字，代表该过程存放的起始地址，是调用指令 CALL 的目标操作数。过程属性有 NEAR 或 FAR 两种。NEAR 属性（近过程）允许过程在段内调用，FAR 属性（远过程）允许过程在段间调用。省略时默认属性为 NEAR。

RET 为过程的返回指令，是过程的出口，但不一定是过程的最后一条指令。一个过程可以有多条 RET 指令，正常情况下至少要执行到一条 RET 指令。

汇编程序根据过程定义语句，当汇编到调用该过程的 CALL 语句时，会自动识别出是段内调用还是段间调用；当汇编到过程中的 RET 时，也会自动识别出是段内返回还是段间返回，从而给出适合的指令代码。

过程定义与调用程序结构举例如下。

```
CODE    SEGMENT
        ASSUME  CS:CODE
MAIN    PROC    FAR             ; 主程序
        …
        CALL    P1              ; 汇编成段内调用
        …
        RET
MAIN    ENDP
P1      PROC    NEAR            ; 定义过程 P1 是 NEAR 过程
        ADD     CX, 1
        RET
P1      ENDP
CODE    ENDS
        END  MAIN
```

一个段可以有多个过程。过程调用还允许嵌套或递归，嵌套或递归的深度只受堆栈的限制。

電子工業出版社

（2）宏定义伪指令

宏的概念与过程很相似。若源程序中经常要用到一个程序段（宏体），可将其定义成一条宏指令（宏定义），于是在源程序中可用这条宏指令语句替代所定义的程序段（宏调用）。当 MASM 汇编程序处理到这条宏指令时，会自动用宏体替换它而扩展成原来的程序段（宏扩展）。

① 宏的定义、调用与扩展。

宏定义格式：　　宏指令名　　MACRO [形参, 形参, …]

…　　；宏体

ENDM

宏调用格式：　　宏指令名　　[实参, 实参, …]

MACRO/ENDM 语句限定所定义的宏。宏指令名代表所定义的宏体的内容，宏一经定义，就像为指令系统增加了一条新的指令一样，在程序中可通过宏指令名对它进行任意次的调用。宏定义中的形参为可选项，若选用了形参，所定义的宏称为带参数的宏。当调用这样的宏时，需用对应的实参去取代，以实现向宏中传递信息。

例如，为了实现 ASCII 与 BCD 码之间的相互转换，往往需要把 AL 中的内容左移 4 位或右移 4 位。若左移 4 位，则可用下列宏定义和宏调用来实现。

```
SHIFT   MACRO                          ; 宏定义
        MOV     CL, 4
        SAL     AL, CL
        ENDM
        …
        SHIFT                          ; 宏调用
        …
```

当汇编程序遇到 SHFIT 这样的调用时，将对应的宏体插入到源程序宏指令语句所在的位置上，来替换这条宏指令并产生目标代码，这就是宏扩展。汇编程序在每一条由宏扩展而产生的指令前冠以“+”。例如：

```
        + MOV   CL, 4                  ; 对应宏指令 SHIFT
        + SAL   AL, CL
```

为了使宏定义适用于不同的情况，可以在宏定义中引入参数。如上例中希望每次用不同的寄存器移位不同的次数，可用下列宏定义和调用来实现。

```
SHIFT   MACRO   X, Y                   ; 带参数的宏定义
        MOV     CL, X
        SAL     Y, CL
        ENDM
        …
        SHIFT   4, AL                  ; 宏调用，AL 左移 4 位。实参与形参按序一一对应
```

② 取消宏指令名伪指令 PURGE。

格式：PURGE　　宏指令名 [, …]

用 MACRO 定义过的宏指令名可用 PURGE 来注销。其他一些宏指令这里不详述。

（3）宏与过程的区别

对程序中需重复使用的一些程序段，既可用宏也可用过程来定义，然后在程序中对它们进行调用。那么，宏与过程又有些什么区别，应如何选用它们呢？总结起来有以下 4 点：

電子工業出版社

① 宏指令语句由汇编程序识别，在程序汇编时完成宏扩展的处理。而调用过程的 CALL 指令语句是在程序执行时完成过程调用的。

② 宏操作可以直接传递参数，而过程不能直接带有参数。当过程之间需要传递参数时，必须通过寄存器、存储器或堆栈来进行。

③ 宏调用只能简化源程序的书写，缩短源程序的长度，并没有缩短其目标代码的长度。汇编程序处理宏指令时，是把宏体插入到宏调用处，对一个宏定义多次调用就要多次插入，所以目标程序占用内存空间并不因宏操作而减少；过程调用却能缩短源程序目标代码的长度，有效地节省内存空间。因为过程在源程序的目标代码中只有一段，无论过程被调用多少次，除了增加 CALL 指令代码，并不增加过程段的代码。

④ 引入宏操作不会在执行目标代码时增加额外的时间开销。相反，过程调用时的 CALL 和 RET 指令涉及保护和恢复现场及断点，这些额外的时间开销会延长目标程序的执行时间。

5．模块定义与通信伪指令

一个任务的汇编语言源程序可由一个或多个模块组成。在多个模块的情况下，每个模块实现一个特定的功能，并且有自己的代码段和数据段等，对应一个*.asm 源文件，是一个独立的汇编单位。当所有模块单独汇编完毕后，再通过 LINK 程序链接成一个完整的可执行程序。MASM 汇编程序提供了模块化编程伪指令，用以实现模块的定义与通信。

（1）模块定义伪指令 NAME/END

它们定义一个可以独立编写及汇编的程序模块。

模块定义格式：

```
NAME    模块名          ; 为模块命名
        …               ; 若干语句
        END     [标号]  ; 结束模块
```

通常，模块命名语句 NAME 可以省略。省略时，模块所在的源程序文件名为模块名。

模块结束语句 END 告诉汇编程序本模块到此结束。如果该模块是程序的主模块或一个任务的独立模块，END 语句中必须有表征程序启动地址的标号。

（2）模块间通信伪指令 PUBLIC/EXTERN

程序的各模块之间必须解决过程或数据的互访和共享问题，使用全局符号说明伪指令 PUBLIC/EXTERN 可以实现这个连接任务。

格式：

```
PUBLIC    名字 [, 名字, …]
EXTERN    名字:类型 [, 名字:类型, …]
```

其中，名字可以是变量名、标号、过程名或由 EQU（或=）伪指令定义的符号名。类型可以是 BYTE/WORD/DWORD（变量）或 NEAR/FAR（过程）和 ABS（由 EQU 定义的常量的属性）。

PUBLIC 说明一个模块内哪些名字为全局的，可以被其他模块引用。EXTERN 说明本模块中使用的哪些名字为外部的（已在其他模块中定义）并已被说明为 PUBLIC 的。各模块间的 PUBLIC/EXTERN 要配对使用，并且指明的类型要一致。

6．条件汇编伪指令

条件汇编伪指令可使汇编程序根据某种条件有选择地对源程序中的某部分语句进行汇编处理。各种条件汇编语句的一般格式如下：

```
        IFxx      条件
```


電子工業出版社

```
        …                           ; 语句体 1
    [ELSE
        …  ]                        ; 语句体 2
    ENDIF
```

根据条件的不同，条件汇编伪指令有 5 种，每种含两条相反条件的伪指令，如表 4-6 所示。

表 4-6　条件汇编伪指令

	格　式	功　能
1	IF　表达式	表达式值非 0，则条件为真，汇编语句体 1
	IFE　表达式	表达式值为 0，则条件为真，汇编语句体 1
2	IFDEF　符号	符号已被定义，则条件为真，汇编语句体 1
	IFNDEF　符号	符号未被定义，则条件为真，汇编语句体 1
3	IFB　<形参>	用在宏定义体。如果宏调用没有用实参替代该形参，则条件为真
	IFNB　<形参>	用在宏定义体。如果宏调用用实参替代该形参，则条件为真
4	IFIDN　<字符串 1>,<字符串 2>	两字符串相同，则条件为真，区别大小写
	IFIDNI　<字符串 1>,<字符串 2>	两字符串相同，则条件为真，不区别大小写
5	IFDIF　<字符串 1>,<字符串 2>	两字符串不同，则条件为真，区别大小写
	IFDIFI　<字符串 1>,<字符串 2>	两字符串不同，则条件为真，不区别大小写

每条 IFxx 伪指令都必须与 ENDIF 配对使用，汇编程序根据要求对条件进行检测：若条件为真，汇编语句体 1 中的语句部分；条件为假，语句体 1 不被汇编。若存在可选的 ELSE 伪指令，语句体 2 中的语句将在条件为假时被汇编。

7．结构定义与使用伪指令

MASM 中还允许用结构（STRUC/ENDS）或记录（RECORD）等伪指令来自定义一些复杂的数据结构，如学籍管理中的学生数据项，包括学生姓名、性别、年龄、成绩等多个不同数据类型的相关字段，可通过这些数据结构对若干相关的单个变量进行整体数据定义。

结构的定义与使用需要三个步骤：结构类型说明→结构变量定义→结构引用。

（1）结构类型说明伪指令 STRUC/ENDS

格式：　结构名　STRUC　　; MASM 6.1x 后用 STRUCT

　　…　　; 结构体，由数据定义语句序列组成

　　结构名　ENDS

例如，将一个学生的姓名、性别、学号、数学成绩、电路成绩等定义为一个结构。

```
STUDENT  STRUC                      ; 结构由 5 个字段共 9 字节构成
         SID      DB  ?             ; 学号
         NAME     DB  5 DUP(")      ; 姓名预留 5 个字符空间
         SEX      DB  'M'           ; 性别
         MATH     DB  0             ; 数学成绩
         CIRCUIT  DB  0             ; 电路成绩
STUDENT  ENDS
```

结构中用数据定义语句 DB、DW 等定义的变量称为字段或结构域，在某些情况下变量本身还可以是另一个结构。

（2）结构变量的定义

结构类型说明只是通知汇编程序增加了一个新的数据类型，但并未给结构分配内存空间，需要通过定义结构变量为它分配内存空间。

電子工業出版社

格式：　[结构变量名]　　结构名　<[域值表]>

其中，结构变量名是用户为定义的结构变量起的名字，可以默认。结构名与结构定义时的结构名相同，不可省略。域值表用于给结构变量的各域赋初值，初值间用逗号分开。初值可以是数值，可以用 DUP 重复操作符，也可以省略，省略时由汇编程序自动给出。

例如：对上例说明的结构 STUDENT，定义若干结构变量并赋初值。

```
        ONE     STUDENT  <  >                      ; 初值为结构说明时的值
        TWO     STUDENT  <02, 'ZHANG', , 87, 90>   ; SEX 字段是结构说明时的值
        THREE   STUDENT  <03, 'WANG', 'F', 78, 92>
        FOUR    TUDENT  5 DUP(<>)     ; 定义有 5 个结构变量的结构数组，初值为结构说明时的值
```

一旦定义结构变量，即为该变量分配内存空间，确定了其在内存中存放的起始地址。结构中各字段具有各自的局部偏移量，是指各字段第一字节与结构起始地址间的字节距离。

（3）结构变量及其字段的引用

引用结构变量，只需直接书写结构变量名；要引用其中的某个字段（也称为成员变量），需要使用专门的结构字段运算符“.”。

格式：结构变量名.结构字段名

例如，对上例定义的各结构变量进行引用，实现从 ONE 到 FOUR[3]、FOUR[4]共 8 个结构变量的 SID 字段分别赋值 1～8。

```
        MOV     ONE.SID, 1
        MOV     SI, OFFSET FOUR       ; 取结构数组首地址→SI
        MOV     AL, 4
        MOV     CX, LENGTH FOUR       ; 取结构数组元素个数 5→CX
INIT:   MOV     [SI].SID, AL          ; 依次给结构数组的 SID 字段赋值
        INC     AL
        ADD     SI, TYPE FOUR         ; SI+9，修改地址指针
        LOOP    INIT
```

4.3.2　80x86/Pentium CPU 扩展伪指令

宏汇编 MASM 5.0 及以上版本针对 80x86/Pentium CPU 的发展增加了许多伪指令，这里仅介绍与选择 CPU 指令集、存储器工作模式及编程模式等相关的部分伪指令，其余结合汇编语言程序设计方法再讨论。

1．处理器选择伪指令

80x86/Pentium 系列 CPU 的指令集在不断地扩充，编程时，需要在程序的开始处用选择伪指令来说明 CPU 所使用的指令系统，如表 4-7 所示，通知汇编程序当前 CPU 的工作方式和使用的指令集。如果省略，则系统默认为 8086/8088 及 8087 指令集。若想使用 32 位寄存器，则必须加上 .386 及以上处理器的选择伪指令。

2．80x86/Pentium CPU 完整段定义的扩充伪指令

（1）完整段定义伪指令

为了表示 32 位 CPU 的功能，80x86/Pentium 完整段定义伪指令如下。

格式：段名　SEGMENT　[定位类型] [组合类型] [字长选择] ['类别名']

　　…　　；段体

　　段名　ENDS

電子工業出版社

表 4-7　处理器选择伪指令

伪指令	功　能	伪指令	功　能
.286/.286C	选择 80286 非特权（实地址模式）指令和 80287 协处理器指令	.286P	选择全部 80286 指令和 80287 协处理器指令
.386/.386C	选择 80386 非特权指令和 80387 协处理器指令	.386P	选择全部 80386 指令和 80387 协处理器指令
.486/.486C	选择 80486 非特权指令	.486P	选择全部 80486 指令
.586/.586C	选择 Pentium 非特权指令	.586P	选择全部 Pentium 指令
.686/.686C	选择 Pentium Pro～Pentium 4 非特权指令	.686P	选择全部 Pentium Pro～Pentium 4 指令
.287	选择 80287 协处理器指令	.387	选择 80387 协处理器指令
.MMX	选择 MMX 指令	.XMM	选择 SSE、SSE2、SSE3、SSE4 指令

其中，增加了针对 32 位 CPU 的“字长选择”项，该项有两种选择。

USE16 — 16 位段模式（对应实地址模式），字长为 16 位，按 16 位方式寻址（段地址 16 位，偏移地址 16 位），最大段长 64 KB。

USE32 — 32 位段模式（对应保护模式），字长为 32 位，按 32 位方式寻址（段选择子 16 位，偏移地址 32 位），最大段长 4 GB。

注意：① 寄存器的位数和段的位数是两个不同的概念，80386 及以上 CPU 无论选择 USE16 还是 USE32，均可使用 32 位寄存器，即 32 位寄存器的使用是由.386、.386p 等伪指令决定的，与 16 位段或 32 位段的选择无关。② 字长选择项省略时默认的段属性为 USE32。因此，在实地址模式编程时，一定要先使用 USE16 说明为 16 位段。③ 选择了 USE16，也可以使用 32 位寄存器，按 16 位方式寻址，这时它们的高 16 位为 0。

（2）80x86/Pentium 汇编语言源程序结构

80x86/Pentium 汇编语言源程序较 8086/8088 源程序在框架结构上的主要差异为：① 开始处增加了处理器选择；② 段定义中有字长选择可选项以确定段模式；③ 允许同时使用代码段 CS、数据段 DS、堆栈段 SS 及附加段 ES、FS 和 GS 共 6 个段。

例如，两个 32 位带符号数的乘法。设在内存的数据段连续存放了两个 32 位（4 字节）的带符号数 DATA1 和 DATA2，编程将它们的 32 位带符号乘积置于 RESULT 的 4 个内存单元中（设结果不超过 32 位）。

```
        .586                                    ; 定义处理器
DATA    SEGMENT  USE16                          ; 数据段按 16 位寻址
DATA1   DD       0FFFFFFBFH                     ; 被乘数
DATA2   DD       0FFFFF000H                     ; 乘数
RESULT  DD       ?                              ; 乘积
DATA    ENDS
STACK   SEGMENT  PARA  STACK  USE16 'STACK'     ; 堆栈段按 16 位寻址
        DB       100 DUP (?)
STACK   ENDS
CODE    SEGMENT  USE16                          ; 代码段按 16 位寻址
        ASSUME  CS:CODE, DS:DATA
START:  MOV      AX, DATA
        MOV      DS, AX
        MOV      EAX, DATA1
        MOV      EBX, DATA2
        IMUL     EAX, EBX                       ; 两个 32 位数相乘→EAX
        JO       ERR                            ; 若结果超过 32 位，OF=1，转 ERR
```


電子工業出版社

```
        MOV     RESULT, EAX                     ; 否则，结果送存
EXIT1:  MOV     AH, 4CH
        INT     21H
ERR:    MOV     DWORD PTR RESULT, 0             ; 结果单元送 0
        JMP     EXIT1
CODE    ENDS
        END START
```

3．存储模式定义伪指令

存储模式定义伪指令用来定义程序中数据和代码的存放格式以及它们占用内存的大小。

格式：.MODEL　　存储模式[, 高级语言]

MASM 有 7 种存储模式可以选择：TINY、SMALL、MEDIUM、COMPACT、LARGE、HUGE、FLAT，含义如表 4-8 所示。[, 高级语言]是可选项，可使用 C、BASIC、FORTRAN、PASCAL 等关键字来指定与哪种高级程序设计语言接口。

表 4-8　内存模式类型

内 存 模 式		说　明
TINY	微型模式	数据和代码放在同一个 64 KB 段内，产生.com 程序。MASM 6.x 支持
SMALL	小模式	程序包含两个段：1 个 64 KB 代码段和 1 个 64 KB 数据段（一般应用程序常用）
MEDIUM	中模式	可以有多个代码段（1 MB 内），但是只有 1 个 64 KB 数据段
COMPACT	紧凑模式	可以有多个数据段（1 MB 内），但是只有 1 个 64 KB 代码段
LARGE	大模式	多个数据段（1 MB 内）与多个代码段（1MB 内），但数据指针不能跨越 64K 段边界
HUGE	巨型模式	多个数据段（1 MB 内）与多个代码段（1 MB 内），数据指针可以跨越段边界
FLAT	平展模式	允许有 1 个 512 KB 的段，用于存储代码和数据。MASM 6.0 支持

MS-DOS 环境下的程序只能选用前 6 种模式，第 7 种模式定义的程序只能在 Windows 9x 等 32 位系统环境中运行。

当独立的汇编语言源程序与高级语言混合编程时，存储模式的选择应该与高级语言的一致。

对于简化段定义，存储模式定义伪指令不能缺少。

4．80x86/Pentium 简化的段定义伪指令

简化的段定义伪指令用来表示一个段的开始，同时说明前一个段的结束。若这个段是程序中的最后一个段，则该段以 END 伪指令结束。具体的简化段定义语句见表 4-9。

表 4-9　简化段定义语句

格　式	功　能	段默认名
.CODE　[段名]	定义代码段。如有多个代码段，要用段名区别	在 TINY、SMALL、COMPACT 及 FLAT 内存模式下，为 _TEXT
.STACK　[长度]	定义堆栈段，并形成 SS、SP 初值。SP=长度，若省略长度，SP=1024	为_STACK
.DATA	定义数据段，要赋初值	为_DATA

使用简化段定义的程序框架如下。此程序将变量名为 ABCD 所指的两个双字进行按位“与”运算，结果存入 EFGP 双字单元。

```
        .MODEL  SMALL                   ; 定义存储模式，指定数据段和代码段允许使用的长度
```


電子工業出版社

```
        .386                            ; 定义处理器方式
        .STACK  64                      ; 定义堆栈段长度为 64 字节，并给 SS、SP 赋初值
        .DATA                           ; 定义数据段
ABCD    DD      11112222H, 0000FFFFH    ; 定义双字数据
EFGP    DD      0                       ; 运算结果缓冲区
        .CODE                           ; 定义代码段
MAIN:   MOV     AX, @DATA               ; @DATA 对应数据段的段地址
        MOV     DS, AX                  ; 建立数据段寄存器 DS 的初值
        MOV     EAX, ABCD               ; 取出第 1 个双字给 EAX
        MOV     EBX, ABCD+4             ; 取出第 2 个双字给 EBX
        AND     EAX, EBX                ; 与运算
        MOV     EFGP, EAX               ; 结果存入 EFGP 中
        MOV     AH, 4CH
        INT     21H                     ; 返回 DOS
        END     MAIN                    ; 整个程序结束
```

SMALL 存储模式下，代码段长度可达 64 KB，数据段和堆栈段为一个组，其总长度可达 64 KB，数据段名@DATA 代表组所对应物理段的段地址。在汇编链接后，程序按代码段（默认 PARA）、数据段（默认 WORD）、堆栈段（默认 PARA）的顺序装入存储区。

注意：① 80386 及以上 CPU 选择伪指令在源程序中的位置不一样可有不同的含义。如果 CPU 选择伪指令放在 .MODEL 伪指令前面，段将被定义成 32 位的段；如果希望 CPU 使用 16 位的段，则应在 .MODEL 伪指令的后面使用 CPU 选择伪指令。② .STARTUP 与 .EXIT 0 也是简化段定义的伪指令。前者表示程序开始，并初始化 DS、SS、SP，程序结束处的 END 伪指令不必再指定程序入口处的标号；后者相当于 MOV AH, 4CH 和 INT 21H 两条指令。据此，上例的 CODE 部分可以改写成：

```
        .CODE                           ; 定义代码段
        .STRATUP                        ; 程序开始
        …
        .EXIT 0                         ; 程序结束，返回 DOS
        END                             ; 整个程序结束
```

4.4 汇编语言程序设计方法

从程序结构上看，汇编语言程序的代码段有顺序、分支、循环、子程序等基本结构，奠定了实现各种复杂程序的基础。本节针对汇编语言的特点，介绍这些基本结构程序的设计方法，并结合实例学习相关的 MASM 6.x 伪指令。

4.4.1 程序设计的基本过程

汇编语言程序可以最大限度地发挥计算机硬件的性能。一般应遵循如下步骤进行设计：

（1）分析问题、建立数学模型并确定算法。在理解和分析问题的基础上，建立数学模型或者总结出若干规律，选择适当的数据结构及合理的算法或解决方案。

（2）绘制程序流程图。流程图可以将算法逻辑、控制结构及数据流程形象地表示出来，是设计思想的直观表现形式。流程图一般由起始框、处理框、判断框、流向线等组成。

電子工業出版社

（3）分配内存空间和寄存器。用指令或伪指令为数据或代码程序分配内存工作单元和寄存器，这是汇编语言程序设计的重要特点之一。

（4）编程并调试。根据流程图和语法规则编写出*.asm 源程序，编制的程序应严格遵守程序结构方面的规定，具有模块化、通用性、简单易读、易于维护等特点，尽可能采用合理的指令和寻址方式，合理地利用 CPU 内部寄存器。

对编制出的程序先做静态语法检查，检查程序是否符合设计思想。再上机并通过调试程序（如 DEBUG）进行动态调试，查找和排除设计错误，直至运行正确。

（5）整理文档。这是程序设计的重要一环，贯穿程序设计的始终。文档包括需求分析、设计方案、程序设计说明、程序使用说明等，文档的编制有利于程序的设计、维护与使用。

汇编语言程序从编制到能够运行需要经过编辑、汇编、链接、调试等过程，如图 4-4 所示。其中，编辑器可以是任一文本编辑环境，编制出的源程序需要保存为 .asm 文件。再用 MASM 宏汇编程序对其进行汇编，形成 .obj 目标文件，同时可获得列表（.lst）文件和映像（.map）文件等。接着，用 LINK 链接程序将一个或几个 .obj 文件链接成一个可执行的 .exe 文件。最后，在调试环境 DEBUG 中进行单步或连续执行，观察内部寄存器、内存的变化，来判断程序是否运行正确。

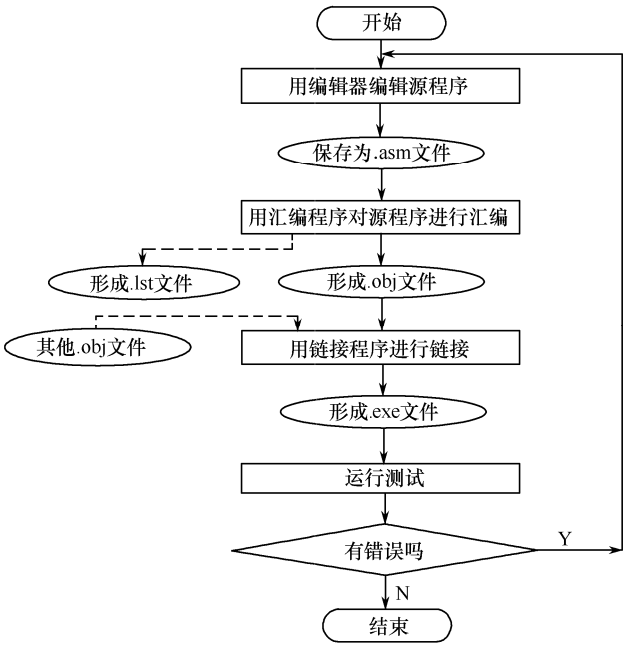


图 4-4　汇编语言程序调试流程

该过程中的每个环节往往需要反复操作，不断完善程序设计，才能达到最终目标。

4.4.2　顺序结构程序设计

顺序结构程序只做直线运行，程序形式简单，任何程序中都包含这种结构。此类程序设计只需依事件发展的先后，选择合适的指令有序地加以组合。

【例 4-2】 内存中自 TABLE 开始的 16 个单元连续存放着 0～15 的平方值（平方表）。任给一数 x（$0 \leqslant x \leqslant 15$）在 XX 单元，查表求 x 的平方值，并将结果存入 YY 单元。

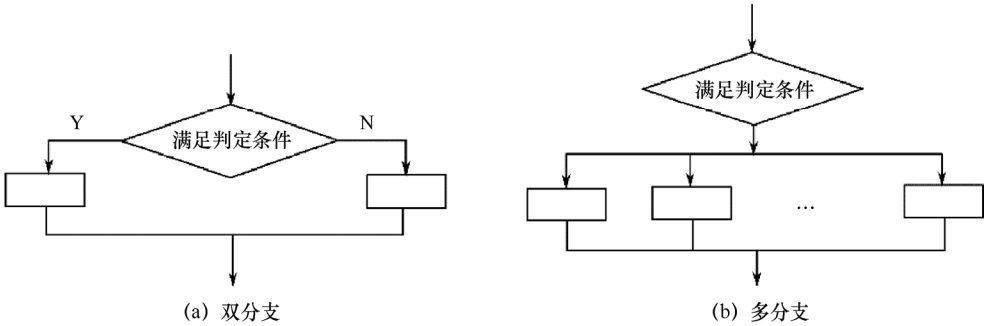
電子工業出版社

分析平方表的存放规律可知，平方表的起始地址与数 x 的和，便是 x 的平方值所在单元的地址。由此设计程序如下：

```
DATA    SEGMENT
TABLE   DB      0,1,4,9,16,25,36,… 121,144,169,196,225    ;定义平方表
XX      DB      ?                                  ;待查数
YY      DB      ?                                  ;待查数的平方
DATA    ENDS
STACK   SEGMENT  PARA  STACK ‘STACK’
        DB      100 DUP (?)               ;定义 100 字节的堆栈空间
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
START   PROC    FAR
        PUSH    DS                        ; 标准序
        MOV     AX, 0
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX                    ; 置数据段寄存器
        MOV     BX, OFFSET  TABLE         ; 置数据指针
        MOV     AH, 0
        MOV     AL, XX                    ; 取待查数
        ADD     BX, AX                    ; 查表
        MOV     AL, [BX]
        MOV     YY, AL                    ; 平方数存 YY 单元
        RET
START   ENDP
CODE    ENDS
        END     START
```

4.4.3 分支结构程序设计

在一个实际问题中，程序始终是直线运行的情况并不多见，通常都会有各种分支。分支结构程序按照给定的条件进行判断，然后根据条件成立或不成立，执行不同的处理过程。分支程序依其支路的多少，可分为双分支和多分支两类，如图 4-5 所示。



(a) 双分支　　(b) 多分支

图 4-5　分支程序结构

電子工業出版社

设计分支程序的关键是设定分支条件。分支结构的实现方法有比较/测试分支和跳转表分支两种。

1．比较/测试分支程序

比较/测试分支程序的设计方法，是在需要分支的地方用比较指令（CMP）或数据操作（ADD、SUB、TEST 等）指令来改变标志寄存器各个标志位（OF、SF、ZF、PF、CF）的值，然后选用合适的条件转移指令 Jcc，测试当前标志位的状态来实现不同的分支转移。这种方法简单明了，适合程序中仅有少数分支的情况。

【例 4-3】 变量 X 的符号函数如下，编写计算函数值 Y 的程序（设 X、Y 均为字节变量，$-128 \leqslant X \leqslant 127$）。

$$Y=\begin{cases}1 & X>0\\0 & X=0\\-1 & X<0\end{cases}$$

这个问题中分支的条件非常简单，只要能够判别 X 的符号，即可得到 Y 的值。将 X 直接与 0 比较，也可以用一条能影响标志位的指令，如“与”“或”等操作，将 X 是否为 0 或 X 的符号反映到 ZF 和 SF 上。算法流程图如图 4-6 所示。

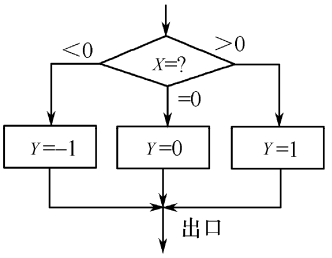


图 4-6　符号函数算法流程图

```
DATA    SEGMENT
XX      DB      X                   ; 存放 X
YY      DB      ?                   ; 存放 Y
DATA    ENDS
STACK   SEGMENT  PARA  STACK 'STACK'
        DB      100 DUP (0)
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX              ; 设置 DS
        MOV     AL, XX              ; 取变量 XX 的值
        CMP     AL, 0               ; X 与 0 两个带符号数比较
        JG      BIGER               ; 带符号数比较的条件转移指令
        JE      FINISH              ; X=0，AL=0
        MOV     AL, -1              ; X<0，-1→AL
        JMP     FINISH
BIGER:  MOV     AL, 1               ; X>0，1→AL
FINISH: MOV     YY, AL              ; 保存函数值→YY
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```

【例 4-4】 将 AL 寄存器低 4 位二进制数以十六进制数形式显示在屏幕上。

键盘输入与屏幕输出都是基于 ASCII 的。因此，本例需要“二进制数→十六进制数→ASCII 值”。“十六进制数→ASCII 值”时需要判断十六进制数在 0～9 之间还是在 A～F 之间。程序段如下：

电子工业出版社

```
        AND     AL, 0FH             ; 取 AL 的低 4 位
        ADD     AL, 30H             ; 十六进制数→ASCII 值的转换
        CMP     AL, 3AH
        JB      PIT                 ; 是 0~9 之间的数，其 ASCII 值加上 30H 即可
        ADD     AL, 07H             ; 是 A~F 之间的数，其 ASCII 值还要加上 7
PIT:    MOV     DL, AL              ; 调用 DOS 系统功能调用 2 号子程序，显示单个字符
        MOV     AH, 02H
        INT     21H
```

2．利用跳转表实现分支程序

如果分支较多，可以在内存中构造一个跳转表，表中或存放这些分支程序的入口地址，或存放跳转到这些分支的指令（每条指令的目标代码长度要一致）。当程序按一定的条件寻址到跳转表中相应的项时，即可使用无条件转移指令 JMP（如 JMP　[BX]）实现分支转移。

【例 4-5】 设有 8 种产品的编号分别为 0，1，2，…，7，各产品的加工子程序名分别为 SBR0，SBR1，…，SBR7。试编写由已知编号转至相应加工子程序处理的程序。

设所有分支为段内分支。将子程序的入口地址连续存放在以 BASE 为首地址（基地址）的跳转表中，如图 4-7 所示。对应任一产品编号，其子程序入口地址在跳转表中的位置满足：

子程序入口地址在跳转表中的位置=表基地址+表内偏移量

=表基地址 BASE+产品编号×2

依据此式可进一步得到子程序入口地址。程序流程图如图 4-8 所示。程序清单如下：

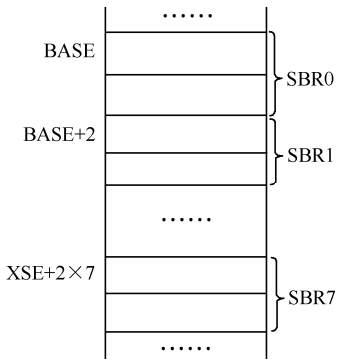


图 4-7　地址跳转表

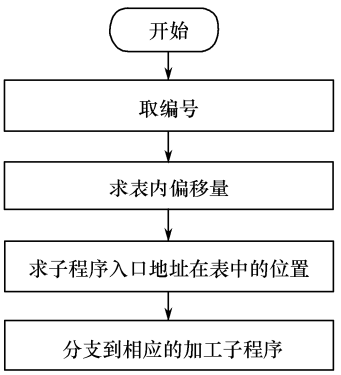


图 4-8　例 4-5 流程图

```
DATA    SEGMENT
BASE    DW  SBR0, SBR1, SBR2, SBR3, SBR4, SBR5, SBR6, SBR7 ; 定义跳转表
BN      DB  ?                                              ; BN 中存放某一产品编号
DATA    ENDS
STACK   SEGMENT  PARA  STACK  'STACK'
        DB       100 DUP (0)
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START   PROC     FAR
        PUSH     DS
        MOV      AX, 0
        PUSH     AX
```


電子工業出版社

```
        MOV     AX, DATA
        MOV     DS, AX
        MOV     BL, BN                          ; 取产品编号
        MOV     BH, 0                           ; 16 位扩展
        SHL     BX, 1                           ; 表内偏移量=产品编号 × 2
        JMP     BASE[BX]                        ; 间接转移到相应的产品加工子程序
SBR0:   …                                       ; 子程序 0
        RET
        …
SBR7:   …                                       ; 子程序 7
        RET
START   ENDP
CODE    ENDS
        END     START
```

若将转移到这些分支程序的指令存放在跳转表中，应定义跳转表在代码段，如图 4-9 及以下程序段所示，BASE 表中每 3 个单元存放一条转移指令（设 JMP 是 3 字节指令）。只要寻址到跳转表中相应的项，即可执行转至某子程序的 JMP 指令，从而实现程序的转移。

```
        MOV     AL, BN          ; 取产品编号
        MOV     AH, 0           ; 高位扩展用
        MOV     BL, AL          ; 表内偏移量=产品编号 × 3
        ADD     AL, AL
        ADD     AL, BL
        MOV     BX, OFFSET  BASE        ; 取跳转表基地址
        ADD     BX, AX          ; 求转移指令在表中的地址
        JMP     BX              ; 间接转移到子程序
        …
BASE:   JMP     SBR0            ; 指令跳转表
        JMP     SBR1
        …
        JMP     SBR7
SBR0:   …                       ; 子程序 0
        …
SBR7:   …                       ; 子程序 7
```

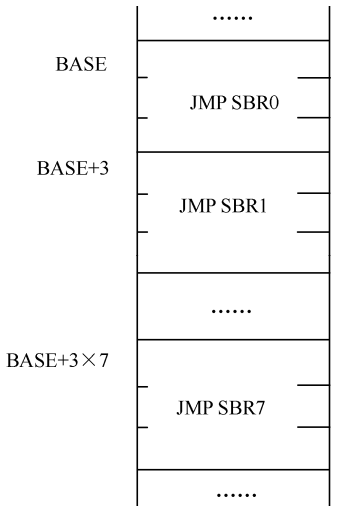


图 4-9　指令跳转表

3．用 MASM 6.x 伪指令设计分支程序

MASM 6.x 引入了.IF、.ELSE 等条件控制伪指令，类似高级语言中分支语句的功能。这些伪指令在汇编时自动生成相应的比较和条件转移指令序列，实现程序分支。利用条件控制伪指令可以简化分支结构编程。

条件控制伪指令的格式如下（方括号内的部分可选）：

```
        .IF 条件表达式              ; 条件为真（值为非 0），执行分支体
            分支体
        [.ELSEIF 条件表达式         ; IF 条件为假（值为 0），且当前 ELSEIF 条件为真，执行分支体
            分支体 ]
        [.ELSE                      ; 前面 IF[以及前面 ELSEIF]条件为假，执行分支体
            分支体 ]
        .ENDIF                      ; 分支结束
```

条件表达式中的操作符如表 4-10 所示。操作符的优先关系为：逻辑非“!”最高，然后是表中左列的比较类操作符，最低的是逻辑与“&&”和逻辑或“||”。“( )”用来改变运算的优先顺序。位测试操作符&的使用格式是“数值表达式 & 数值”，相当于执行 TEST 指令和相应的条件转移指令。

表 4-10　条件表达式中的操作符

操 作 符	功　能	操 作 符	功　能	操 作 符	功　能
= =	等于	&&	逻辑与	CARRY?	CF=1？
! =	不等于	||	逻辑或	OVERFLOW?	OF=1？
>	大于	!	逻辑非	PARITY?	PF=1？
> =	大于等于	&	位测试	SIGN?	SF=1？
<	小于	( )	改变优先级	ZERO?	ZF=1？
< =	小于等于	—			

条件表达式中的变量，若是用 DB、DW、DD 定义的，则按无符号数处理。若需要进行有符号数的比较，相应变量应使用 SBYTE、SWORD、SDWORD 伪指令来定义。采用寄存器或常数作为条件表达式的数值参加比较时，默认也是无符号数。如果要作为有符号数，可以用 SBYTE PTR 或 SWORD PTR 操作符来指明。

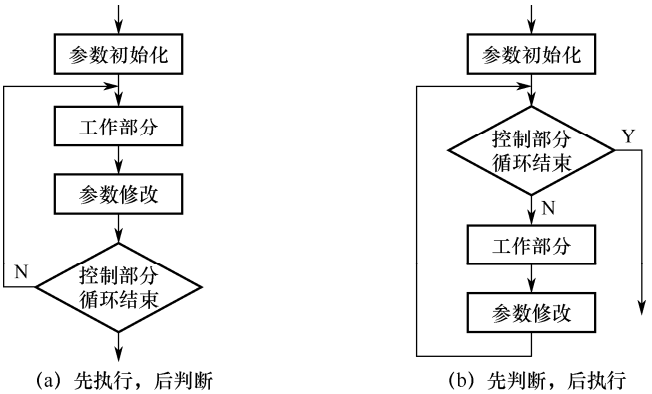
例如，用条件控制伪指令实现例 4-3 中求符号函数 Y 的值。

```
        MOV     AL, XX                  ; XX 由 SBYTE 定义为带符号数
.IF     SBYTE  PTR   AL>0               ; 将 XX 与 0 比较
        MOV     YY, 1                   ; 第一分支体：条件满足，1→YY
.ELSEIF SBYTE  PTR  AL=0
        MOV     YY, 0                   ; 第二分支体：条件满足，0→YY
.ELSE
        MOV     YY, -1                  ; 第三分支体：条件不满足，-1→YY
.ENDIF
```

4.4.4　循环结构程序设计

1．循环程序的结构与循环控制方法

循环程序是在满足某些条件时对一段程序的重复执行，一般由 4 部分组成，如图 4-10 所示。



(a) 先执行，后判断　　(b) 先判断，后执行

图 4-10　循环程序结构

電子工業出版社

① 参数初始化。设置循环次数计数器，设定各变量的初值或变量地址指针的初值等，为循环体正常工作准备必要的条件。

② 工作部分。需要重复执行的程序代码，这是循环程序的核心。

③ 参数修改。修改循环计数器及各变量地址等，保证循环程序在循环时得到下一次参加运算的正确数据。

④ 循环控制。依据给定的循环次数或循环条件，决定是否结束循环。

工作部分、参数修改及循环控制构成了一个循环体，参数初始化部分位于循环体之外。循环体各部分有时可以简化，形成互相包含或交叉的情况，不一定总是分得很明显。

循环程序有“先执行，后判断”和“先判断，后执行”两种基本结构形式。前者至少执行一次循环体，在循环条件已知时常选用这种结构。后者的特点是进入循环后，先判断循环结束的条件，如果一进入循环就满足循环结束条件，则循环体一次也不执行。在循环次数未知，需要根据特定条件来控制循环时常选用这种结构。

循环程序的运行与结束依赖于循环控制方法，常用方法如下。

① 计数控制：循环次数已知，每循环一次，循环计数器都需要进行加或减的调整。

② 条件/状态控制：循环次数未知，在执行循环时，需要通过判定某种条件或状态的真假来控制循环。

当循环控制方法的可能选择方案不止一种时，需要分析，选择效率较高的一种方法。

【例 4-6】 统计字节数据块-1，-3，5，6，9，…中负元素的个数。

统计负数个数的方法之一是查看每个数的符号位并统计符号位为 1 的数的个数。这种重复性工作可用循环程序实现，循环控制条件是循环次数（数据块的长度）。程序设计如下：

```
        .MODEL   SMALL
        .STACK   256                    ; 定义堆栈段,长度为 256 字节
        .DATA
BUF     DB       -1, -3, 5, 6, 9, …     ; 定义若干字节带符号数
CUNT    EQU      $-BUF                  ; 计算字节数据块的长度
RESULT  DW       ?                      ; 定义存放结果单元
        .CODE
        .STARTUP
        MOV      BX, OFFSET  BUF        ; 建立数据指针
        MOV      CX, CUNT               ; 设置循环次数
        MOV      DX, 0                  ; 置结果初值
LP1:    MOV      AL, [BX]               ; 取数据
        AND      AL, AL                 ; 影响标志位
        JNS      PLUS                   ; 是正数，转去 PLUS
        INC      DX                     ; 是负数，负数个数+1
PLUS:   INC      BX                     ; 调整指针
        LOOP     LP1                    ; (CX-1)≠0，继续循环
        MOV      RESULT, DX             ; 存入负数个数
        .EXIT 0
        END
```

【例4-7】AX寄存器中有一个16位二进制数，编程统计其中1的个数，结果存放在CX中。

这个程序最好采用“先判断，后执行”的结构。如果 AX 中的 16 位全为 0，则不必再做统计工作。相应程序段如下：

電子工業出版社

```
        MOV     CX, 0                   ; 置结果计数器初值
LP1:    AND     AX, AX                  ; AX=0?
        JZ      LP2                     ; 是，退出循环
        SAL     AX, 1                   ; 否，AX 的最高位移至 CF 中
        JNC     ZERO                    ; CF=0，转 ZERO 继续循环
        INC     CX                      ; CF=1，结果计数器 CX 加 1
ZERO:   JMP     LP1
LP2:    …
```

本例中，循环的结束可以用计数值 16 来控制，但使用 AX 为全 0 这个特征条件可以提前结束循环，缩短了程序的运行时间，提高了效率。

2．多重循环程序设计

循环程序有单循环和多重循环两种。多重循环程序设计与单循环程序设计的方法是一致的，应分别考虑各重循环的控制条件及程序实现，相互之间不能混淆。

【例 4-8】 软件延时程序。

程序中每条指令都有一定的执行时间，因而利用软件可以实现延时。当要求延时的时间较长时，可采用多重循环。

```
SOFTDLY PROC                            ; 指令执行时间
        MOV     BL, 10                  ; 4T
DELAY:  MOV     CX, 2801                ; 内循环延时 10ms  4T
WAIT1:  LOOP    WAIT1                   ; 17T/5T
        DEC     BL                      ; 3T
        JNZ     DELAY                   ; 16T/4T
        RET                             ; 20T
SOFTDLY ENDP
```

在这个双重循环中，内循环次数 CX 由 2801 减至 0，BL 维持不变，大约可实现 10 ms 的延时。外循环进行 10 次，可实现 100 ms 的延时（设 CPU 时钟周期 T=210 μs）。

延时时间 t 计算如下：

$$t = \underbrace{\{4+[10\times(\overbrace{(4+(2801\times 17-12))}^{\text{内循环}}+3+16)-12]+20\}}_{\text{外循环}}\times T$$

3．用 MASM 6.x 伪指令设计循环程序

MASM 6.x 提供了循环控制伪指令来设计循环程序，可以简化编程，使程序结构清晰。这些循环控制伪指令包括：用于先执行后判断结构的.REPEAT 和.UNTIL 及.REPEAT 和.UNTILCXZ，用于先判断后执行结构的.WHILE 和.EDNW，还有分别表示无条件退出循环和转向循环体开始处的 .BREAK、.CONTINUE 等。主要伪指令的使用形式如表 4-11 所示。其中，条件表达式与条件控制伪指令 .if 语句中的条件表达式规定是一样的。

【例 4-9】 求 1～100 数字之和，结果存入 AX 中。

（1）使用 UNTIL 结构，程序段为：

```
        XOR     AX, AX
        MOV     CX, 100
        .REPEAT
            ADD     AX, CX
```


電子工業出版社

表 4-11　MASM 6.x 循环控制伪指令使用形式

使用　.REPEAT 和.UNTIL	使用　.WHILE 和.EDNW
.REPEAT　　；重复执行循环体 　　循环体 .UNTIL　条件表达式　　；直到条件为真 .REPEAT　　；重复执行循环体 　　循环体 .UNTILCXZ　[条件表达式] 　　；CX←CX−1 直到 CX=0 或条件为真	.WHILE　条件表达式　　；条件为真，执行循环体 　　循环体 .EDNW　　；循环体结束

```
            DEC     CX
        .UNTIL  CX == 0
或      XOR     AX, AX                  ; 100, 99, …, 2, 1 倒序累加
        MOV     CX, 100
        .REPEAT
            ADD     AX, CX
        .UNTILCXZ                       ; CX←CX-1，直到 CX=0
```

（2）使用 WHILE 结构，循环体可以编写为：

```
        .WHILE  CX != 0
            ADD  AX, CX                 ; 从 100, 99, …, 2, 1 倒序累加
            DEC  CX
        .ENDW
```

【例 4-10】 改写例 4-6“统计字节数据块−1，−3，5，6，9，…中负元素的个数”的程序，且当遇到第一个非负数时终止统计。相应程序段为：

```
        .DATA
BUF     SBYTE   −1, −3, 5, 6, 9, …      ; 定义若干字节带符号数
        …
        .STARTUP
        MOV     BX, OFFSET  BUF         ; 建立数据指针
        MOV     CX, CUNT                ; 设置循环次数
        MOV     DX, 0                   ; 置结果初值
        .REPEAT
            .IF  SBYTE  PTR[BX]<0       ; 是负数，DX 加 1
                INC     DX
            .ELSE                       ; 是非负数，终止统计
                .BREAK
            .ENDIF
                INC     BX
        .UNTILCXZ
        MOV     RESULT, DX              ; 结果值存入 RESULT
```

4.4.5　子程序设计与调用技术

使用子程序（过程）是程序设计的一种重要方法。常常将功能相对独立或需要多次重复使用的程序段定义为子程序，以有效地简化源程序结构。需要强调的是，中断服务程序是一种特殊的子程序（过程）。因为中断的发生是随机的、不可预料的，所以对中断的服务只能编制成过程的形式。当中断发生时，转去它完成相应的功能并返回断点。

電子工業出版社

1．子程序的调用与返回

子程序在代码段中以过程形式存在，对子程序的调用方式有段内调用、段间调用、直接调用和间接调用之分。普通子程序的调用与返回由 CALL 和 RET 指令实现，中断服务程序由 INT 指令或外部事件触发调用，由 IRET 指令返回。

子程序调用实际上就是程序的转移过程，与转移指令不同的是，这个转移需要返回。以普通子程序为例，主程序中的 CALL 指令在执行时，先将断点地址（CALL 指令的下一条指令地址：段内调用是 IP，段间调用是 IP 与 CS）进堆栈保护，再将 CALL 中指定的子程序的入口地址送到 IP 或 IP 与 CS，将控制转移到子程序。子程序执行完毕后，通过 RET 指令返回。RET 指令从堆栈栈顶取出断点地址重新装入 IP（段内返回）或 IP 与 CS（段间返回），使控制回到断点处继续执行。

2．子程序设计与应用中的问题

（1）现场的保护与恢复

在 CPU 将控制转移到子程序前，主程序已使用的某些寄存器或存储单元中的信息，可能在子程序运行后还要继续使用，必须先将它们压入堆栈加以保护，待子程序结束后再将其恢复，这通常称为现场的保护与恢复。

保护与恢复现场的典型格式，是将 PUSH 和 POP 指令成对地安排在子程序的开始和结束，如形式一。也可安排在主程序中 CALL 指令的前后，如形式二。可根据需要选用。

形式一：

```
DTOB    PROC
        PUSH    BX
        PUSH    AX
        …
        POP     AX
        POP     BX
        RET
DTOB ENDP
```

形式二：

```
        …
        PUSH    BX
        PUSH    AX
        CALL    子程序
        POP     AX
        POP     BX
        …
```

特别地，对中断服务程序，一定要在服务程序中安排保护和恢复指令。因为中断是随机出现的，主程序转入中断服务程序的位置是不固定的，无法在主程序中安排这一段指令。

（2）参数传递

参数是主程序和子程序之间的数据通道。一般将子程序从主程序获取的数据称为入口参数（输入参数），子程序返回给主程序的数据称为出口参数（输出参数）。参数的具体内容可以是数据本身（传数值），也可以是数据的存储地址（传地址）。

参数传递的方法主要是利用一些公共区域，如寄存器、存储单元或堆栈。主程序和子程序要按约定配合，一方将参数送到哪儿，另一方就到哪里去取。

電子工業出版社

（3）子程序说明

子程序具有共享性，可被其他程序调用，因此设计子程序时应有必要的注释说明，包括：功能描述（含子程序的名称、功能、性能指标等）、入口和出口参数、占用的寄存器和存储单元。子程序又可以调用其他子程序或实例等。

（4）调用子程序的若干技术

① 子程序嵌套调用。子程序中含有对其他子程序的调用就是嵌套，只要堆栈空间允许，嵌套层次不限。

② 子程序递归调用。子程序直接或间接调用自身时称为递归调用，含有递归调用的子程序称为递归子程序。

③ 可重入子程序。子程序被调用后，还未执行完又被另一个程序调用的过程称为重入，能够重入的子程序称为可重入子程序。子程序的重入不同于子程序的递归，重入是被动的进入，递归是主动的进入，重入的调用之间往往没有关系，递归调用之间却是密切相关的。递归子程序也是可重入子程序。遗憾的是，DOS 功能调用是不可重入的。

3．子程序应用举例

【例 4-11】寄存器传递参数。求字节数组 ARRAY 中所有元素之和，结果送至 SUM 字单元。

使用寄存器传递参数是指，在通用寄存器中存放主程序和子程序之间需要传递的参数。这种方法简单快捷，但因 CPU 中寄存器数量有限，仅适合于参数较少的情况。

本例，我们在主程序中设置地址和计数初值，求数组元素和的功能由子程序来完成。

```
STACK   SEGMENT  PARA  STACK 'STACK'
        DB       100 DUP(?)
STACK   ENDS
DATA    SEGMENT
ARRAY   DB       d1, d2, d3, …, dn        ; 定义字节数组
COUNT   EQU      $-ARRAY
SUM     DW       ?                        ; 设和为一个字
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
START:  MOV      AX, DATA
        MOV      DS, AX
        LEA      SI, ARRAY  }             ; 入口参数准备，将需要传递的参数送入寄存器
        MOV      CX, COUNT  }
        CALL     SUM1                     ; 调用子程序 SUM1 求和，返回值在 AX 中
        MOV      SUM, AX                  ; 和存于 SUM 单元
        MOV      AH, 4CH                  ; 返回 DOS
        INT      21H
        ; 子程序名：SUM1。程序功能：求字节数组和。
        ; 入口参数：SI=数组首址，CX=数组长度。出口参数：AX=数组和。使用寄存器：AX，CX，SI。
SUM1    PROC     NEAR                     ; 子程序（过程）开始
        MOV      AX, 0                    ; 数组和通过 AX 寄存器回送到主程序
        CMP      CX, 0
        JZ       EXIT1
AGAIN:  ADD      AL, [SI]                 ; 求数组元素的和
```


電子工業出版社

```
        ADC     AH, 0
        INC     SI
        LOOP    AGAIN
EXIT1:  RET
SUM1    ENDP                            ; 子程序（过程）结束
CODE    ENDS                            ; 代码段结束
        END     START                   ; 整个程序结束
```

【例4-12】利用存储单元传递参数。求字数组之和（假定和不溢出），结果送至SUM字单元。

利用存储单元传递参数的方法分为两种：① 直接存储单元传递，利用约定的存储单元直接传递数据本身，这与寄存器传递参数类似。② 参数地址表传递，定义一个表存放待传参数的地址（称为参数地址表），将参数地址表的首地址通过寄存器带进子程序，则子程序可从参数地址表中取得所需参数的地址，继而取得参数。本例使用方法②。

```
STACK   SEGMENT  PARA  STACK 'STACK'
        DB       100 DUP(?)
STACK   ENDS
DATA    SEGMENT
ARRAY   DW       d1, d2, d3, …, dn          ; 定义字数组
COUNT   DW       N                          ; 定义字数组中元素个数
SUM     DW       ?                          ; 设和为一个字
TABLE   DW       3 DUP(?)                   ; 定义参数地址表 TABLE
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV      AX, DATA
        MOV      DS,AX
        MOV      TABLE, OFFSET  ARRAY
        MOV      TABLE+2, OFFSET  COUNT  ; 参数地址送入地址表 TABLE
        MOV      TABLE+4, OFFSET  SUM
        LEA      BX, TABLE                  ; 地址表的首地址→BX
        CALL     PROADD                     ; 调用子程序 PROADD 求和并存储
        MOV      AH, 4CH
        INT      21H                        ; 返回 DOS
; 子程序名：PROADD。程序功能：求字数组和。使用寄存器：AX、CX、BP、SI、DI
; 入口参数：BX=地址表首地址，参数地址在地址表中。出口参数：和在 SUM 单元中
PROADD  PROC     NEAR
        PUSH     AX                         ; 保护现场 AX、CX、BP、SI、DI
        …
        PUSH     DI
        MOV      SI, [BX]                   ; 数组 ARRAY 首地址→SI
        MOV      BP, [BX+2]                 ; 数组长度单元 COUNT 地址→BP
        MOV      CX, DS:[BP]                ; 数组长度→CX
        MOV      DI, [BX+4]                 ; 存储和单元 SUM 地址→DI
        MOV      AX, 0
ADDIT:  ADD      AX, [SI]                   ; 循环求和
        ADD      SI, 2
        LOOP     ADDIT
        MOV      [DI], AX                   ; 存储和
```


電子工業出版社

```
        POP     DI                          ; 恢复现场
        …
        POP     AX
        RET
PROADD  ENDP
CODE    ENDS
        END     START
```

【例 4-13】 利用堆栈传递参数。改写例 4-12 为堆栈传递参数，并将求和子程序置于另一个代码段。

堆栈传递参数的方法是：将主、子程序中需要传递的参数压入堆栈，使用时再从堆栈中取出。本例程序执行过程中堆栈的变化如图 4-11 所示。

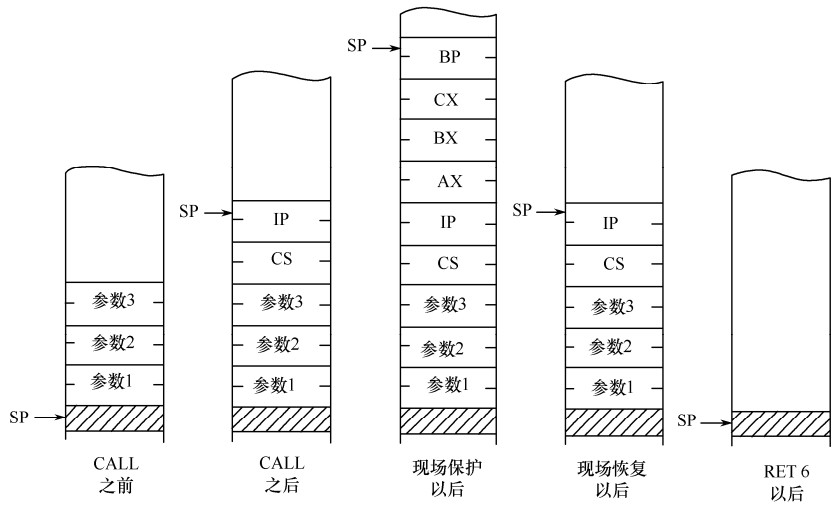


图 4-11　堆栈的变化

```
STACK   SEGMENT  PARA  STACK 'STACK'
        DB       100 DUP (?)
STACK   ENDS
DATA    SEGMENT
ARRAY   DW       d1, d2, d3, …, dn          ; 定义字数组
COUNT   DW       N                          ; 定义字数组中元素个数
SUM     DW       ?                          ; 设和为一个字
DATA    ENDS
EXTERN  PROADD:FAR                          ; 主程序和子程序 PROADD 在两个不同的段中
CODE1   SEGMENT
        ASSUME   CS:CODE1, DS:DATA
START:  MOV      AX, DATA                   ; 主程序开始
        MOV      DS, AX
        LEA      BX, ARRAY                  ; 地址参数 1 进栈
        PUSH     BX
        LEA      BX, COUNT                  ; 地址参数 2 进栈
        PUSH     BX
        LEA      BX, SUM                ; 地址参数 3 进栈
        PUSH     BX
```


電子工業出版社

```
        CALL    FAR PTR PROADD          ; 段间调用，求和
        MOV     AH, 4CH
        INT     21H
CODE1   ENDS
        END     START
; 子程序在另一个代码段中
PUBLIC  PROADD
CODE2   SEGMENT
        ASSUME  CS:CODE2
;子程序名: PROADD。程序功能:数组求和。使用寄存器:AX、BX、CX、BP、SI
;入口参数: 数组、数组长度及存和单元地址在栈中。出口参数: 和在 SUM 字单元
PROADD  PROC    FAR                     ; 子程序类型定义为 FAR
        PUSH    AX                      ; 保护现场 AX、BX、CX、BP
        …
        MOV     BP, SP                  ; 堆栈操作需要用 BP 间接寻址
        MOV     BX, [BP+14]             ; 取得地址参数
        MOV     CX, [BX]
        MOV     BX, [BP+12]
        MOV     SI, [BP+16]
        MOV     AX, 0
ADDIT:  ADD     AX, [SI]                ; 求和
        ADD     SI, 2
        LOOP    ADDIT
        MOV     [BX], AX                ; 保存和
        …                               ; 恢复现场 BP、CX、BX、AX
        POP     AX
        RET     6                       ; 返回并废除 3 个地址参数
PROADD  ENDP
CODE2   ENDS
        END                             ; END 语句中不需标号
```

利用堆栈传递参数需要注意，子程序调用并保护现场后（见图 4-11），被传参数位于“高地址”区，被断点地址和现场值“覆盖”，这时不能直接用 POP 指令弹出。可先使 BP 指向栈顶，以 BP 加位移量的形式指向栈中参数所在单元，再用 MOV 指令取出参数。

4．扩展的带参数过程定义伪指令

MASM 6.x 参照高级语言的函数形式扩展了 PROC 伪指令的功能，使其具有带参数的能力，方便了过程间参数的传递。带参数的过程定义伪指令格式如下，各参数说明如表 4-12 所示。

```
过程名  PROC    [调用距离] [语言类型] [作用范围] [<起始参数>][USES 寄存器列表]
                [, 参数:[类型]] …
        LOCAL   变量名[个数][, 类型][, …]      ; 过程中使用的局部变量表
        …                                      ; 汇编语言语句
过程名  ENDP
```

过程定义中的 LOCAL 伪指令用来说明过程中使用的局部变量，其中可选的“[个数]”表示同类型数据的个数。被 LOCAL 伪指令说明的局部变量由汇编系统自动存于堆栈。

使用带参数的 PROC 来定义过程，还需要配套的过程声明伪指令 PROTO 和过程调用伪指令 INVOKE（而不是 CALL）。MASM 6.x 规定 PROTO 与 INVOKE 的格式如下：

電子工業出版社

表 4-12　过程定义的选项参数说明

参　数	说　明
调用距离	NEAR 或 FAR，表示该过程是近或远调用。简化段定义中，默认值由.MODEL 语句选择的存储模式决定
语言类型	可以是任何有效的语言类型，确定该过程采用的命名约定和调用约定。如果选用 C 语言，约定名字前加下画线，参数传递顺序从右到左，且参数的长度可变
作用范围	可以是 PUBLIC（默认）、PRIVATE、EXPORT，表示该过程是否对其他模块可见
起始参数	采用这个格式的 PROC 伪指令，汇编系统将自动创建过程的起始代码和结束代码，用于传递堆栈参数以及清除堆栈等。起始参数表示传送给起始代码的参数，有多个时用逗号分开
寄存器列表	指通用寄存器名，多个时用空格分隔。对 USES 中罗列的寄存器，汇编系统将自动在起始代码及结束代码中产生相应的入/出栈指令
参数:类型	表示该过程使用的形式参数及其类型。参数类型可以是任何 MASM 有效的类型。在 16 位段中默认的类型是 WORD，32 位段中默认的是 DWORD

```
过程名    PROTO    [调用距离][语言类型][, 参数, [类型]]…
          INVOKE   过程名[, 参数…]
```

过程在调用前必须声明，声明伪指令 PROTO 中各项必须与相应的过程定义伪指令 PROC 中各项一致。过程调用伪指令 INVOKE 自动创建调用过程所需要的代码序列，其中的参数是通过堆栈传递给过程的实参，它们可以是数值表达式、寄存器对、ADD 标号（ADDR 标号传送的是标号的地址：WORD 类型时是偏移地址，DWORD 类型时是段地址和偏移地址）。

【例 4-14】 运用过程声明和过程调用编写上述例 4-11。

```
          .MODEL SMALL
SUM       PROTO  C, COUNTP:WORD, ARRAYP:WORD     ; 声明过程
          .STACK
          .DATA
COUNT     EQU      10                            ; 数组的元素个数
ARRAY     DB       12H, 25H, 0F0H, 0A3H, …       ; 数组
RESULT    DW       ?                             ; 和
          .CODE
          .STARTUP
          INVOKE   SUM, COUNT, OFFSET ARRAY      ; 调用过程
          MOV      RESULT, AX                    ; 保存和
          .EXIT 0
; 子程序名: SUM。程序功能: 字节数组求和
; 入口参数: COUNTP=数组长度, ARRAYP=数组首地址。出口参数: AX=数组和
SUM       PROC  C  USES  BX  CX,COUNTP:WORD,ARRAYP:WORD
          MOV      BX, ARRAYP                    ; 数组的偏移地址→BX
          MOV      CX, COUNTP                    ; 数组的元素个数→CX
          XOR      AX, AX
SUMD:     ADD      AL, [BX]                      ; 求和
          ADC      AH, 0
          INC      BX
          LOOP     SUMD
          RET
SUM       ENDP
          END
```


電子工業出版社

5．BIOS/DOS 功能程序调用

在80x86/Pentium微机系统的ROM中固化有一组外部设备驱动与管理软件，组成了IBM PC 的基本输入/输出系统，称为 ROM BIOS，它处于系统软件的底层。DOS 在此基础上开发了一组输入/输出设备处理程序 ibmbio.com，这也是 DOS 与 ROM BIOS 的接口。在 ibmbio.com 的基础上，DOS 还开发了文件管理等一系列处理程序 ibmdos.com。DOS 的命令处理程序 command.com 与 ibmbio.com、ibmdos.com 这两种程序就构成了基本 DOS 系统。

在汇编语言程序设计中，用户可通过使用 ROM BIOS 和 DOS 系统提供的这些功能模块程序，来编制直接管理和控制计算机硬件设备的软件，如完成基本 I/O 设备（显示器、键盘、打印机、硬盘等）、内存和文件的管理，以及中断向量、时钟和日历的读/写等操作，以扩充汇编语言的功能。

（1）调用 BIOS/DOS 功能程序的基本方法

BIOS/DOS 的所有功能程序并非用子程序名来区别，调用时不需使用 CALL 指令，也不必关心其内部结构和细节，以及所使用的设备的物理特性、接口方式等，只要采用如下统一格式的三步操作，配合一条软中断指令 INT　n 即可，这称为中断调用。

① 依据功能程序对入口参数的要求，传送入口参数到指定寄存器。

② 将子功能编号（也称为功能调用号）送 AH 寄存器。

③ 执行软中断指令 INT　n。

软中断指令 INT　n 中的中断类型码 n 表征调用不同的功能程序（集）。ROM BIOS 中的功能程序对应 n=05H～lFH，DOS 中的的功能程序对应 n=20H～3FH 等。有的软中断指令对应一个分支程序，调用时不需上述步骤②；有的软中断指令则对应一个分支程序集，如 INT　10H 对应近 20 个分支程序，INT　21H 对应 100 多个分支程序等，调用时必须将分支（子）功能编号送入 AH。常用的 BIOS 与 DOS 功能调用说明参看附录 B（http://www.hxedu.com 网上注册下载）。

【例 4-15】 将一个 ASCII 字符显示在屏幕当前光标位置。

使用 ROM BIOS 中的 n=10H、功能调用号是 0EH 的程序段如下：

```
MOV     AL, '?'          ; 要显示的字符送入 AL
MOV     AH, 0EH          ; 功能调用号送入 AH
INT     10H              ; 调用 10H 软中断
```

有的功能程序不需要入口参数，这时①可略去。如使当前程序返回 DOS 状态的程序段：

```
MOV     AH, 4CH
INT     21H              ; 调用 21H 软中断对应的 4CH 号子功能，返回 DOS
```

（2）DOS 功能程序与 DOS 系统功能调用

DOS 功能程序（n=20H～3FH 等）在更高层次上提供了部分与 BIOS 类同的功能，使用起来很方便。与 BIOS 的功能程序调用相比，两者的区别如下：① 调用 BIOS 功能程序比调用 DOS 功能程序要复杂，但运行速度快，功能更强；② DOS 功能调用只在 DOS 环境下适用，而 BIOS 功能调用不受任何操作系统的约束；③ 某些功能只有 BIOS 才具有。

在 DOS 功能程序调用中，通过 INT　21H 软中断指令来实现的调用被称为 DOS 系统功能调用。DOS 系统功能调用体现了 DOS 的核心功能，对应的 100 多个子功能包括：磁盘的读/写及控制管理、内存管理和基本输入/输出管理等，并已全部顺序编号。

现介绍 DOS 系统功能调用中常用的输入/输出子功能。

電子工業出版社

① 单字符输入（01H、07H 功能）。01H 功能调用等待从键盘输入一个字符，当按下一个键时，调用返回并将该字符的 ASCII 值送入 AL，同时在屏幕上显示该字符。若按下的键是 Ctrl-Break，则执行 INT　23H。例如：

```
        MOV     AH, 01H          ; 01H 功能调用不需要入口参数，返回值在 AL 中
        INT     21H              ; AL 为输入字符的 ASCII 值
```

07H 功能调用同 01H，等待从键盘输入一个字符并将该字符的 ASCII 值送入 AL，但不在屏幕上回显，也不检测 Ctrl-Break 键。

② 单字符输出（02H 和 05H 功能）。将寄存器 DL 中的单个字符输出到显示器（02H 功能）或打印机（05H 功能）上。例如：

```
        MOV     DL, 'A'          ; 字符 A 的 ASCII 值送入入口参数 DL
        MOV     AH, 2
        INT     21H              ; 屏幕显示字符'A'
```

③ 单字符输入或输出（06H 功能）。06H 功能调用可以从键盘输入字符，或向屏幕输出字符，并且不检查 Ctrl-Break 键。执行 06H 功能调用时，CPU 不等待用户按键。

入口参数 DL=FFH 时，表示从键盘输入。若当前标志 ZF=0，表示有键按下且 AL 中为键入字符的 ASCII 值；若标志 ZF=1，表示无键按下，AL 中的值与键值无关。

入口参数 DL≠FFH 时，表示向屏幕输出，这时 DL 中为输出字符的 ASCII 值。例如：

```
        MOV     DL, 0FFH         ; 从键盘输入字符
        MOV     AH, 6
        INT     21H

或      MOV     DL, 24H          ; 将字符$输出到屏幕上
        MOV     AH, 6
        INT     21H
```

④ 字符串输入（0AH 功能）。等待从键盘输入字符串，将其存入以 DS:DX 为首址的内存缓冲区，同时显示该字符串，输入过程以回车键结束。

调用前，要求事先在数据段定义一个输入缓冲区，并使 DS:DX 指向缓冲区首址（入口参数）。缓冲区的第 1 字节指出缓冲区能容纳的最大字符数（1～255，不能为 0），第 2 字节用于存放实际输入的字符数（回车键除外），这个数在调用返回时由系统自动填入。用户从键盘键入的字符从第 3 字节开始存放，直到用户输入回车键为止，并将回车符（0DH）加在键入字符串的末尾。因此，设置的缓冲区最大长度应比希望输入的最多字符数多 1 字节。

⑤ 字符串输出显示（09H 功能）。将一个以$字符结尾的字符串（不包括$）输出到显示器。欲显示的字符串需事先存入内存缓冲区，并使 DS:DX 指向它的首地址（入口参数）。

【例 4-16】 利用 DOS 系统功能调用实现人机对话。根据屏幕上显示的提示信息，从键盘输入字符串并存入内存缓冲区。

```
DATA    SEGMENT
BUF     DB      100                     ; 输入缓冲区最大长度=最多键入字符个数+1
        DB      ?                       ; 存放实际输入字符个数
        DB      100 DUP (0)             ; 输入字符缓冲区（最多输入 99 个字符）
MESG    DB      'WHAT IS YOUR NAME?', 0DH, 0AH, '$'      ; 欲显示的提示信息
DATA    ENDS
CODE    SEGMENT
```


電子工業出版社

```
        ASSUME   CS:CODE, DS:DATA
START:  MOV      AX, DATA
        MOV      DS, AX
        …
        MOV      DX, OFFSET  MESG          ; DS:DX 指向 MESG
        MOV      AH, 9                     ; 屏幕显示提示信息
        INT      21H
        MOV      DX, OFFSET  BUF           ; DS:DX 指向 BUF
        MOV      AH, 10                    ; 接收键盘输入
        INT      21H
        …
CODE    ENDS
        END      START
```

4.5 模块化程序设计技术

良好的程序结构应该是模块化的，程序分段和子程序等都是使程序模块化的基本方法。但是，在开发较大或复杂的程序时，还需要专门的模块化程序设计技术。

4.5.1 模块化程序设计的特点与规范

1. 模块化程序设计的主要特点

模块化程序设计是指将一个较大的“任务”分解成若干具有独立功能的“子任务”，每个子任务命名为一个模块，对每个模块单独编辑和编译，生成各自的源文件 .asm 和 .obj 文件，然后由 LINK 程序将各模块有效地链接在一起，形成一个完整的可执行文件 .exe。

采用模块化程序设计技术具有以下特点：

- 可以将程序分配给多人编写，以缩短程序设计周期。
- 有利于程序的编写、调试、修改与更新，提高软件设计质量。
- 便于多种程序语言的联合使用。
- 模块程序可放在库里供多个程序使用。

2. 模块化程序设计规范

模块化程序设计中的主要问题是模块划分、模块设计及模块间的链接等，一般规范如下：

- 模块的独立性要强且大小划分适中，每个模块的功能要明确、单一。
- 模块内部的聚合性要好。聚合性体现了模块的专一性和统一性的程度，聚合性好说明模块内部结构紧凑，整体性好，独立性强。
- 每个模块最好只有一个入口和一个出口，这样既有利于程序调试，也不易出错。
- 模块间的关系要明确。划分模块是一个自顶向下的设计过程。在主模块确定后，其余各模块最好再分层，形成树状层次结构。各层间是单向依赖关系，每个模块只归其上一级模块或同层模块调用。
- 将结构化程序设计思想应用于模块，使模块程序由基本结构（顺序、分支和循环等）组合或嵌套而成，程序设计中还可结合子程序等的使用。

電子工業出版社

4.5.2 程序中模块间的关系

具体进行模块化程序设计时，应处理好程序中各模块间的链接关系，主要包括：全局符号的定义与引用、模块间的转移、各模块的组合形式及模块间的通信等。

1．全局符号的定义与引用

单个模块中使用的符号（变量、标号或子程序名等）为局部符号。一个模块中定义的符号如不另加说明，则均为局部符号，局部符号只能在定义它的模块中使用。

多个模块可共同使用的符号为全局符号。模块化编程伪指令表明，只要将局部符号在定义和使用它的模块中分别用 PUBLIC 和 EXTERN 语句说明，即可作为全局符号（又称外部符号）使用，全局符号构成了模块间通信的主要渠道。

2．模块间的转移

模块间的转移有两种：近（段内）转移和远（段间）转移。它们都是通过转移指令来实现的，具体的转移指令是 JMP、CALL 和 INT。

3．多个模块的组合形式

一个源程序的每个模块常常都有自己的数据段、代码段和堆栈段，当 LINK 程序对所有模块进行链接形成一个可执行文件时，模块中的各个段有多种组合链接关系，这可通过段定义伪指令 SEGMENT 中的属性项“组合类型”进行选择，如 4.3 节中表 4-5 所示，再次引用如表 4-13 所示。

表 4-13　各种组合类型表示的链接关系

组合类型	含　义
NONE（默认状态）	本段与其他模块中的同名段无逻辑关系，不组合，各自有自己的段起始地址
PUBLIC	在满足定位类型的前提下，LINK 程序将本段与其他模块中说明为 PUBLIC 的同名段邻接在一起，共用一个段地址，即合成一个物理段
STACK	LINK 程序将所有堆栈段链接成一个连续段，组合后的物理段长度等于参与组合的各堆栈段的长度和
COMMON	各模块中由 COMMON 方式说明的同名段重叠覆盖，有着相同的起始地址。重叠部分的内容取决于参与覆盖的最后一个段的内容，复合段的长度等于参与覆盖的最长段的长度
MEMORY	本段定位在所有链接在一起的其他段的最后（存储器高地址区域）。若有多个 MEMORY 段，汇编程序认为所遇到的第一个为 MEMORY，其余为 COMMON

4.5.3 模块化程序设计举例

【例 4-17】 求无序表中的最大元素及其位置。

为了说明模块化程序设计思想，将本程序分成两个模块来编写：模块 A 为主程序，负责一些基本条件的设置，如无序表的起始地址以及其中的数据个数；求最大值及其位置的过程放在子模块 B 中。程序如下：

```
; 模块 A，文件名 MAIN.ASM
EXTERN   FOUND: NEAR                          ; 引用外部符号
SSEG     SEGMENT  PARA  STACK 'STACK'
         DB       100 DUP (?)
SSEG     ENDS
DATA1    SEGMENT
```


電子工業出版社

```
ARRAY   DB      d1, d2, d3, …, dn       ; 定义带符号数的字节表
COUNT   EQU     $-ARRAY                 ; 数据个数
DATA1   ENDS
CODE    SEGMENT  WORD  PUBLIC 'CODE'
        ASSUME  CS:CODE, DS:DATA1
MAIN:   MOV     AX, DATA1
        MOV     DS, AX                  ; 装入段地址
        MOV     CX, COUNT               ; 装入数据个数
        LEA     SI, ARRAY               ; 设置 SI 为表的起始地址
        CALL    FOUND                   ; 段内调用，找出最大元素及位置
        MOV     AH, 4CH                 ; 返回 DOS
        INT     21H
CODE    ENDS
        END     MAIN
; 模块 B，文件名 SUB.ASM
PUBLIC  FOUND                           ; 说明为全局符号
SSEG    SEGMENT  PARA  STACK ‘STACK’
        DB      200 DUP(?)
SSEG    ENDS
DATA2   SEGMENT
MAX     DB      ?
PLACE   DB      ?
DATA2   ENDS
CODE    SEGMENT  WORD  PUBLIC 'CODE'
        ASSUME  CS:CODE                 ; 未重新指定 DS，DS 保持不变
FOUND   PROC  NEAR
        MOV     DH, 1                   ; 参加比较的两个数中后一个数据的位置
        MOV     DL, 0                   ; 参加比较的两个数中前一个数据的位置
        DEC     CX                      ; N 个数，比较 N-1 次
        MOV     AL, [SI]                ; 第一个数据→AL
COMP:   CMP     AL, [SI+1]              ; AL 与后一个数据比较
        JGE     BIGGER                  ; 前一个 >= 后一个，AL 中就是当前最大值
        MOV     AL, [SI+1]              ; 当前最大值→AL
        MOV     DL, DH                  ; DL 记录当前最大值的位置
BIGGER: INC     SI
        INC     DH
        LOOP    COMP                    ; 循环比较，直至 CX=0
        ASSUME  DS:DATA2
        MOV     BX, DATA2               ; 设置 DS 为 DATA2 的段地址
        MOV     DS, BX
        MOV     MAX, AL                 ; 存最大值在 DATA2 的 MAX 单元
        MOV     PLACE, DL               ; 存最大值位置在 DATA2 的 PLACE 单元
        RET
FOUND   ENDP
CODE    ENDS
        END                             ; END 标志本模块的结束，不需标号
```

说明：

① 两个模块中的堆栈段段名相同，以 STACK 为组合类型，汇编与链接后将组成一个

電子工業出版社

统一的长度为 300 字节的堆栈段。两个模块有各自独立的数据段（组合类型默认），但代码段被合并成为一个段（都是 PUBLIC），因此模块 A 中的主程序对模块 B 中过程 FOUND 的调用实际上是一个近（段内）调用。

② 模块 B 中的过程 FOUND 要被模块 A 引用，因此在模块 B 中被说明成全局的，在模块 A 中被说明成外部的。

③ 在模块 B 中求最大值时，用的是模块 A 中数据段定义的数据，所以在模块 B 的开始处，不需设置 DS 的值，而求得的最大值结果存放在模块 B 的数据段。

④ 主程序在模块 A 中，因此该模块的 END 伪指令使用了标号，用来指定整个程序执行的入口点为 MAIN 标号所指的指令。

⑤ 可用 MASM 命令对程序 MAIN.ASM 和 SUB.ASM 分别进行编译，格式如下：

```
D: > MASM  MAIN   ↙
D: > MASM  SUB    ↙
```

再用 LINK 命令将它们链接成文件名为 MAIN 的可执行文件 MAIN.EXE：

```
D: > LINK  MAIN.OBJ+SUB.OBJ   ↙
```

4.6 综合应用程序设计举例

80x86/Pentium CPU 有多种工作模式。8086/8088 的工作模式是实模式，也是 80286～Pentium 兼容 8086/8088 并复位后默认的工作模式；通过设置相应的寄存器，80286～Pentium 可进入保护模式，充分发挥 CPU 的全部性能；如果希望在保护模式下运行一个或多个 16 位的程序，可将系统切换到虚拟 8086 模式。本节进一步说明实模式与保护模式下的编程方法。

4.6.1 16 位实模式程序设计

实模式的默认条件下，MASM 只接受 8086/8088 的 16 位指令集。其编程的主要特点是，只能使用 8 位或 16 位寄存器，对内存寻址只能使用 SI、DI 及 BX、BP 寄存器。

【例 4-18】 多位非压缩的 BCD 数算术运算。

内存区从 FIRST 和 SECOND 开始的单元中分别存放两个 4 位非压缩 BCD 数，数据存放的规则是，低位在低地址，高位在高地址。求这两个数的和并存放在 THIRD 开始的内存单元。两个 4 位的 BCD 数相加，最高位可能产生进位。为了避免丢失这个进位，给加数、被加数及结果存放单元都开辟 5 个字节单元。程序代码如下：

```
DATA    SEGMENT
FIRST   DB      4, 7, 9, 2, 0           ; 十进制数 02974
SECOND  DB      6, 2, 3, 7, 0           ; 十进制数 07326
THIRD   DB      5 DUP(0)                ; 存放结果
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA, ES:DATA
UNPACK: MOV     AX, DATA
        MOV     DS, AX              ; DS 指向数据段
        MOV     ES, AX              ; ES 指向数据段
        LEA     SI, FIRST
```


電子工業出版社

```
        LEA     BX, SECOND
        LEA     DI, THIRD
        MOV     CX, 4+1                 ; 进行 5 次加法
        CLD
RETRY:  LODSB
        ADD     AL, [BX]                ; 对应的 1 位进行加法运算
        AAA
        STOSB                           ; 保存当前值
        INC     BX
        ADC     BYTE PTR[BX], 0         ; 进位值加到加数的下一位
        LOOP    RETRY
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     UNPACK
```

【例 4-19】 BCD 码转换为二进制数。

将 AX 中的 4 位 BCD 码转换为二进制数，结果仍存于 AX 中。

BCD 码用 4 位二进制数表示 1 位十进制数的编码，所以 AX=9827H 表示值为 9827D。4 位 BCD 码 $D_3D_2D_1D_0$（0～9999）对应的二进制数应为：

$$(((D_4\times10+D_3)\times10+D_2)\times10+D_1)\times10+D_0 \quad （设 D_4=0）$$

显然，完成转换需操作 4 次“×10 后再取数累加”，因此设计如下含有 4 次循环的程序。

```
W10     DW      10                      ; 十进制数权值
; 子程序名: BTO2。入口参数: AX=BCD 码。出口参数: AX=二进制数。
BTO2    PROC  NEAR
        …                               ; 保护现场
        MOV     BX, AX                  ; 保存 AX 中的 BCD 码到 BX
        MOV     AX, 0                   ; 结果单元清 0
        MOV     CX, 4                   ; 共处理 4 位 BCD 码
RETRY:  PUSH    CX
        MOV     CL, 4
        ROL     BX, CL                  ; 1 位 BCD 码左移到 BX 中的低半字节
        POP     CX
        MUL     W10                     ; AX×10→DX:AX
        PUSH    BX                      ; 保存 BX 当前值
        AND     BX, 000FH               ; 取出 BX 中的 1 位 BCD 码
        ADD     AX, BX                  ; 累加到 AX 中
        POP     BX                      ; 恢复 BX
        LOOP    RETRY                   ; 4 次循环未做完，继续
        …                               ; 恢复现场
        RET
BTO2    ENDP
```

【例 4-20】 二进制数转换为 BCD 码。

将 AX 中的二进制数转换成 4 位 BCD 码，转换的结果仍存于 AX 中（设 AX 中的数值小于十进制数 10000）。

算法：将 AX 中的二进制数除以 1000 得到的商是千位上的 BCD 码，所得余数除以 100

電子工業出版社

得到的商是百位上的 BCD 码，所得余数再除以 10 得到的商是十位上的 BCD 码，最后所得的余数是个位上的 BCD 码。显然，完成转换需要操作 4 次“被除数÷权值后再整合”。

```
W1000    DW       1000, 100, 10, 1          ; 十进制数千、百、十、个位的权值
; 子程序名: AX2TOB。入口参数: AX=二进制数。出口参数: AX=压缩 BCD 码。
AX2TOB   PROC  NEAR
         …                                  ; 保护现场
         XOR     BX, BX                     ; 存放 BCD 码的临时单元清 0
         MOV     SI, OFFSET  W1000          ; 权值所在首地址送 SI
         MOV     CX, 4                      ; 循环次数 4→CX
RETRY:   PUSH    CX
         MOV     CL, 4
         SHL     BX, CL                     ; 先转换高位 BCD 码
         MOV     DX, 0                      ; DX:AX 组成被除数
         DIV     WORD PTR [SI]              ; 除以权值，商→AX、余数→DX
         OR      BX, AX                     ; BX 中为压缩 BCD 码
         MOV     AX, DX                     ; 余数送 AX
         POP     CX
         ADD     SI, 2                      ; 地址加 2，指向下一个权值
         LOOP    RETRY                      ; 连续转换 4 次
         MOV     AX, BX                     ; BCD 码由 BX→AX
         …                                  ; 恢复现场
         RET
AX2TOB   ENDP
```

二进制数转换成 BCD 码的另一种算法是“连续除以 10 取余”：首先将 AX 中的数除以 10，所得余数为个位上的 BCD 数；所得的商再作为被除数来除以 10，得到的余数为十位上的 BCD 数……直到被除数小于 10 时，得到最高位的 BCD 数。可据此另编制程序实现这个转换。

【例 4-21】 从键盘输入字符。若是回车键，则退出程序；若是小写字母，则显示对应的大写字母，否则直接显示。

A～Z 字符对应的 ASCII 值为 41H～5AH，a～z 字符对应的 ASCII 值为 61H～7AH。所以，要将小写字母转换为大写字母，只需将其 ASCII 值减去 20H 即可。程序设计如下：

```
         .MODEL   SMALL
         STACK
         .CODE
         .STARTUP
START:   MOV      AH, 01H              ; 等待从键盘上输入一个字符
         INT      21H                  ; 按下键的 ASCII 值存放在 AL 中
         CMP      AL, 0DH              ; 判断按下的键是否为回车键
         JZ       LP2                  ; 是，转移到 LP2，退出程序
         CMP      AL, 61H              ; 不是，与'a'比较
         JC       LP1                  ; 若 AL<'a'，则转去直接送显
         CMP      AL, 7BH              ; 与'z'比较
         JNC      LP1                  ; 若 AL>'z'，则转去直接送显
         SUB      AL, 20H              ; 否则在'a'~'z'之间，减去 20H，变为大写字母
LP1:     MOV      DL, AL               ; DL=欲显示字符的 ASCII 值
         MOV      AH, 02H              ; 设置 02H 号功能，显示字符
```


電子工業出版社

```
        INT     21H
        JMP     START                   ; 重新读取新的键值、判断…
LP2:    .EXIT 0
        END
```

【例 4-22】 在字节数组中找出第一个非零元素，并显示它的下标。

```
        .MODEL  SMALL
        .STACK
        .DATA
ARRAY   DB      0, 0, 0, 25H, 0, 0, 34H, 36H, 45H
COUNT   EQU     9
        .CODE
        .STARTUP
        MOV     CX, COUNT
        MOV     DI, -1                  ; DI 为元素下标
NEXT:   INC     DI                      ; 从 0 号元素起，逐个与 0 比较
        CMP     ARRAY[DI], 0
        LOOPZ   NEXT
        JNE     OK                      ; ZF=0，即有非零元素，转 OK
        MOV     DL, '0'                 ; 没有找到非零元素，显示一个'0'
        JMP     SHOW
OK:     MOV     DX, DI
        OR      DL, 30H                 ; 将下标转换成相应的 ASCII 值
SHOW:   MOV     AH, 02H
        INT     21H                     ; 显示下标
        .EXIT 0
        END
```

【例 4-23】 冒泡法给无序表中的元素排序。

排序的算法有很多，冒泡排序是最常用的一种。其方法是：对无序表从前至后，两两元素比较，逆序则交换。设从地址 ARRAY 开始的缓冲区中有一个字数组，要使该数据表中的 N 个带符号数按照从小到大的次序排列，用冒泡算法实现的过程如下。

从第 1 个数开始依次进行相邻两个数的比较，即第 1 个数与第 2 个数比，第 2 个与第 3 个比……若两个数的次序是顺序（符合排序要求），则不做任何操作；若是逆序，就交换这两个数的位置。经过一遍全表扫描的 $N-1$ 次比较后，最大的数排到了表的最后。用同样的方法再进行第二遍扫描，这时只需考虑前 $N-1$ 个数之间的 $N-2$ 次比较，扫描完毕，次大的数排到了后面……以此类推，在进行了 $N-1$ 遍的扫描比较后则完成了排序。

冒泡法最大可能的扫描遍数为 $N-1$。但是，往往有的数据表在第 I 遍（$I<N-1$）扫描后可能已经成序。为了避免后面不必要的扫描比较，可在程序中引入一个交换标志。若在某一遍扫描比较中一次交换也未发生，则表示数据已按序排列，在这遍扫描结束时就停止程序循环，结束排序过程。用双重循环实现的冒泡算法程序清单如下：

```
        .MODEL  SMALL
        .STACK
        .DATA
ARRAY   DW      d1, d2, d3, …, dn       ; 定义带符号数的字数组
COUNT   EQU     ($-ARRAY)/2             ; 表中的数据个数
FLAG    DB      -1                      ; 交换标志，初值为-1
```


電子工業出版社

```
        .CODE
        .STARTUP
SORT:   MOV     BX, COUNT               ; 元素个数→BX
LP1:    CMP     FLAG, 0                 ; 数组已有序?
        JE      EXIT1                   ; 是，排序结束
        DEC     BX                      ; 否，置本遍扫描比较（内循环）次数
        MOV     CX, BX                  ; 内循环次数→CX
        MOV     SI, 0                   ; 置数组的偏移地址
        MOV     FLAG, 0                 ; 预置交换标志 FLAG 为 0
LP2:    MOV     AX, ARRAY[SI]           ; 取一个数据→AX
        CMP     AX, ARRAY[SI+2]         ; 与下一个数比较
        JLE     NEXT                    ; 前一个数<=后一个数，顺序，转 NEXT
        XCHG    AX, ARRAY[SI+2]         ; 逆序，交换两个数
        MOV     ARRAY[SI], AX
        MOV     FLAG, -1                ; 置交换标志 FLAG 为-1
NEXT:   ADD     SI, 2                   ; 修改地址指针
        LOOP    LP2                     ; 内循环进行两两数据的比较
        JMP     LP1                     ; 内循环结束，继续下一遍排序
EXIT1:  .EXIT 0                         ; 排序完成，返回 DOS
        END
```

4.6.2 基于 32 位指令的实模式程序设计

如果针对 80386 及以上 CPU 来编写基于 32 位指令的实模式程序，先要用 CPU 选择伪指令来说明所使用的 32 位 CPU，并以选择 USE16 段模式（完整段定义）或在 CPU 选择伪指令之前加 .MODEL 伪指令（简化段定义）来明确是在实模式下编写程序。这时可以使用所有的 8、16 和 32 位寄存器，32 位操作数，且所有 32 位寄存器都可用于内存寻址（按 16 位方式在 1MB 内存空间中寻址，分段最大 64 KB）。

【例 4-24】 将一个 64 位数据逻辑左移 8 位。

```
        .386                                ; 选择处理器
DATA    SEGMENT  USE16                      ; 数据段以 16 位寻址
QVAR    DQ       1234567887654321H          ; 定义 8 字节变量
DATA    ENDS
CODE    SEGMENT  USE16                      ; 代码段以 16 位寻址
        ASSUME   CS:CODE, DS:DATA
SHIFT:  MOV      AX, DATA
        MOV      DS, AX
        MOV      ESI, OFFSET QVAR
        MOV      EAX, DWORD PTR [ESI]       ; 取低 32 位
        MOV      EDX, DWORD PTR [ESI+4]     ; 取高 32 位
        MOV      CX, 8                      ; 设置循环次数
AGAIN:  SHL      EAX, 1                     ; 低 32 位逻辑左移
        RCL      EDX, 1                     ; 高 32 位带进位循环左移
        LOOP     AGAIN                      ; 不足 8 次继续移位
        MOV      DWORD PTR QVAR, EAX        ; 保存结果低 32 位
        MOV      DWORD PTR QVAR[4], EDX     ; 保存结果高 32 位
        MOV      AH, 4CH
        INT      21H
```


電子工業出版社

```
CODE    ENDS
        END     SHIFT
```

【例 4-25】 采用冒泡法，将 ESI 指定的缓冲区中 ECX 个 32 位带符号数按从大到小排序。

```
        .MODEL  SMALL
        .386                                    ; 16 位段
        .STACK
        .DATA
COUNT   EUQ  10
DARRAY  DD  20, 4500H, 3F40H, -1, 7F000080H, 81000000H, 0FFFFFFF1H, -45000011, 12345678, 87654321
        .CODE
        .STRATUP                                ; 程序开始
        XOR     ESI, ESI
        MOV     SI, OFFSET DARRAY
        MOV     ECX, COUNT-1
OUTLP:  MOV     EDX, 0                          ; 外循环
INLP:   CMP     EDX, ECX                        ; 内循环
        JAE     BOTM
        MOV     EAX, [ESI+EDX*4+4]
        CMP     [ESI+EDX*4], EAX                ; 比较前后两个数据的大小
        JGE     NSWAP
        XCHG    [ESI+EDX*4], EAX
        MOV     [ESI+EDX*4+4], EAX
NSWAP:  INC     EDX
        JMP     INLP
BOTM:   LOOP    OUTLP
        .EXIT 0                                 ; 程序结束，返回
        END
```

4.6.3 基于多媒体指令的实模式程序设计

只有 Pentium MMX 及以上的 CPU 才支持 MMX、SSE~SSE4 等多媒体指令。其中，MMX 指令集只能处理紧缩整型数据，而 SSE～SSE4 指令集则提供了紧缩浮点型数据的并行处理能力。这里仅介绍 MMX 指令的编程应用。

【例 4-26】 8 个元素（每个元素 16 位，1 个字）的向量点积计算程序设计，即求

$$a \times b = a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3 + \cdots + a_8 \times b_8$$

```
        .MODEL SMALL          ; 本例需使用 MASM 6.11 以上版本，在 Pentium 及以上的机器上运行
        .586
        .MMX
        .STACK
        .DATA
A_VER   DQ    0001000200030004H, 0005000600070008H     ; 给出 a[i]元素（共 8 个字）
B_VER   DQ    0001000200030004H, 0005000600070008H; 给出 b[i]元素（共 8 个字）
AB_VER  DD    ?                                    ; 存放结果（32 位）
NOMSG   DB    'MMX NOT FOUND', 0DH, 0AH, '$'
YESMSG  DB    'MMX FOUND', 0DH, 0AH, '$'
        .CODE
```


電子工業出版社

```
        .STARTUP
        MOV     EAX, 1                          ; 判断 MMX CPU 是否存在
        CPUID                                   ; 取 CPU 的 ID 标识码
        TEST    EDX, 00800000H                  ; D23=1 表示有 MMX 功能
        JNZ     MMX_F                           ; 有 MMX 功能，转去 MMX_F
        MOV     AH, 9                           ; 提示 CPU 不支持 MMX 功能
        MOV     DX, OFFSET NOMSG
        INT     21H
; 以下利用整数指令实现元素的积之和运算
        MOV     SI, OFFSET A_VER
        MOV     DI, OFFSET B_VER
        MOV     CX, 8                           ; 8 次乘法(不支持 MMX 时使用 16 位乘)
START1: MOV     AX, [SI]                        ; 取元素 a[i]
        IMUL    WORD PTR [DI]                   ; 与 b[i]相乘，乘积在 DX、AX 中
        PUSH    DX
        PUSH    AX
        ADD     SI, 2
        ADD     DI, 2
        LOOP    START1
        MOV     CX, 7                           ; 7 次加法
        POP     EAX
START2: POP     EDX                             ; 取乘积
        ADD     EAX, EDX                        ; 求和
        LOOP    START2
        MOV     AB_VER, EAX                     ; 存结果
        JMP     START4
MMX_F:  MOV     AH, 9                           ; 提示 CPU 支持 MMX 功能
        MOV     DX, OFFSET YESMSG
        INT     21H
; 以下采用 MMX 指令实现元素的积之和运算，MM0 中为：a4×b4+a3×b3 及 a2×b2+a1×b1
;                                     MM1 中为：a8×b8+a7×b7 及 a6×b6+a5×b5
        MOVQ    MM0, A_VER                      ; 取元素 a[i]前 4 个（64 位）到 MM0
        PMADDWD MM0, B_VER                      ; 与元素 b[i]前 4 个相乘，再加
        MOVQ    MM1, A_VER+8                    ; 取元素 a[i]后 4 个（64 位）到 MM1
        PMADDWD MM1, B_VER+8                    ; 与元素 b[i]后 4 个相乘，再加
; 组合结果
        PADDD   MM0, MM1                        ; MM0+MM1→MM0（64 位）
        MOVQ    MM1, MM0                        ; 复制 MM0 到 MM1（64 位 t）
        PSRLQ   MM1, 32                         ; 将 MM1 右移 32 位
        PADDD   MM0, MM1                        ; MM0+MM1→MM0（32 位）
        MOVD    AB_VER, MM0                     ; 存 32 位结果值
        EMMS                                    ; 退出 MMX 状态
START4: .EXIT 0
        END
```

注意：MMX 寄存器是使用浮点数据寄存器实现的，编程时需将 MMX 程序与浮点处理程序置于各自独立的代码段中，不要在指令级混用 MMX 指令和浮点指令。而 SIMD 浮点寄存器 XMM7～XMM0 是独立设计的，程序设计时可以混用 SSE 指令与 80x87 浮点指令。

電子工業出版社

4.6.4 保护模式程序设计

保护模式支持多任务。基于保护模式的保护机制，在设计保护模式程序时需要掌握两点：① 对系统中的多个不同任务，将每个任务放在不同的虚拟地址空间；② 对同一任务，依其代码和数据的信任度与重要性的不同为之分配不同的特权级。

保护模式和实模式以同样的方法使用和执行指令。开机或复位时 CPU 总是进入实模式，实模式具有特权级 0。从实模式切换到保护模式的一般步骤如下，反向操作则可回到实模式。

① 利用伪指令构造系统进入保护模式工作的各种表，如全局/中断描述符表 GDT/IDT、任务状态段 TSS 和各任务所涉及的堆栈等。

② 初始化全局描述符表寄存器 GDTR 和中断描述符表寄存器 IDTR 等。

③ 打开 A20 地址线。

④ 通过指令使 CR0 的 D0（PE）置 1，切换到保护模式。

⑤ 根据保护模式，设置各段的选择子。

实模式和保护模式的相互切换是保护模式编程的要点，以下举例说明。

【例 4-27】 以十六进制数的形式显示从内存地址 110000H 开始的 256 字节的值。

本例指定 110000H 内存区域，说明有必要切换到保护模式，因为在实模式下不能访问该内存区域。按照上述步骤本例应该：① 定义相关结构与宏，做好切换到保护模式的准备（在保护模式下需通过描述符表数据结构实现对内存的访问）；② 切换到保护模式；③ 将指定内存区内容传送到常规内存区；④ 切换回实模式；⑤ 使用 INT 21H 显示缓冲区内容。具体如下：

```
; 定义段描述符的结构
DESC          STRUC                  ; 每个段描述符占 8 字节
LIMITL        DW  0                  ; 段界限（BIT0～BIT15）
BASEL         DW  0                  ; 段基地址的低 16 位（BIT0～BIT15）
BASEM         DB  0                  ; 段基地址的 16～23 位（BIT16～BIT23）
ATTRIBUTES    DB  0                  ; 段属性（取值 98H 或 92H）
LIMITH        DB  0                  ; 段界限 BIT16～19（含段属性的高 4 位）
BASEH         DB  0                  ; 段基地址的 24～31 位（BIT24～BIT31）
DESC          ENDS
; 定义伪描述符（用于装入全局或中断描述符表寄存器）
PDESC         STRUC                  ; 装入 48 位 GDTR 时要用到
LIMIT         DW  0                  ; 16 位段界限
BASE          DD  0                  ; 32 位段基地址
PDESC         ENDS
;定义打开 A20 地址线的宏（保护模式下访问的地址空间超过 1MB，要用到 A20 地址线）
ENABLE_A20        MACRO              ; 通过 92H 端口的输出，使地址线 A20 打开
         PUSH     AX
         IN       AL, 92H
         OR       AL, 00000010B      ; 在 92H 端口的 D1 位输出 1，打开 A20 地址线
         OUT      92H, AL
         POP      AX
ENDM
; 定义将 CR0 寄存器中的 D0 置 1 的宏，以便切换到保护模式
TO_PROTECT        MACRO
```


電子工業出版社

```
        MOV     EAX, CR0
        OR      EAX, 1
        MOV     CR0, EAX
ENDM
;定义一个段间跳转的宏指令，以保证在进入保护模式时将代码段的选择子装入 CS 中
JUMP16          MACRO   SELECTOR, OFFSETS
        DB      0EAH                    ; 0EAH 为 JUMP 指令的机器码
        DW      OFFSETS                 ; 16 位段偏移量
        DW      SELECTOR                ; 段选择子
ENDM
```

从保护模式返回到实模式时，只要做和以上相反的动作即可。

```
; 定义一个回到实模式的宏（将 CR0 寄存器中的 D0 清 0）
TO_REAL         MACRO
        MOV     EAX, CR0
        AND     EAX, 0FFFFFFFEH
        MOV     CR0, EAX
ENDM
; 用一个段间跳转的宏，清指令预取队列，使 CS 内容为代码段地址
; 关闭 A20 地址线
DISABLE_A20     MACRO
        PUSH    AX
        IN      AL, 92H
        AND     AL, 0FDH                ; 在 92H 端口的 D1 位输出 0，关闭 A20 地址线
        OUT     92H, AL
        POP     AX
ENDM                                    ; 恢复 DS，使其内容为数据段地址
```

基于以上定义，设计保护模式下的 0 级代码段 PCODE 来实现数据传送，完整程序如下：

```
        .386P                           ; 使用 386 全部指令集
DATA    SEGMENT USE16                   ; 定义 16 位数据段
GDT     LABEL   BYTE                    ; 定义全局描述符表 GDT
DUMMY   DESC<>                          ; 空描述符
PSEG    DESC<0FFFFH, PCODE, , 98H, ,>   ; 保护模式下的代码段描述符
PCODE_SEL =PSEG - GDT                   ; 保护模式下代码段描述符的选择子
DSEGS   DESC<0FFFFH, 0, 11H, 92H, ,>    ; 源数据段描述符，段基地址 110000H
DSEGS_SEL = DSEGS - GDT                 ; 源数据段描述符的选择子
DSEGD   DESC<0FFFFH,  DATA, , 92H, ,>   ; 目标数据段描述符
DSEGD_SEL = DSEGD-GDT                   ; 目标数据段描述符的选择子
GDTLEN = $-GDT                          ; 全局描述符表长度
VGDTR   PDESC<GDTLEN-1, >               ; 伪描述符，界限为 GDT 的实际长度
BUFFER  DB  256 DUP (0)                 ; 256 字节的目标缓冲区
DATAE   NDS
CODE    SEGMENT USE16                   ; 定义 16 位代码段
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     BX, 16                  ; 准备要加载到 GDTR 的伪描述符
        MUL     BX                      ; 现在是实模式，所以段地址要左移 4 位
```


電子工業出版社

```
        ADD     AX, OFFSET GDT                  ; 加上 GDT 表的偏移地址
        ADC     DX, 0                           ; 得到 GDT 的 32 位段基地址 DX、AX
        MOV     WORD PTR VGDTR.BASE, AX         ; 设置 VGDTR 的 BASE 内容
        MOV     WORD PTR VGDTR.BASE+2, DX
        MOV     AX, PSEG.BASEL                  ; 初始化代码段描述符
        MUL     BX
        MOV     PSEG.BASEL, AX                  ; 代码段开始偏移为 0
        MOV     PSEG.BASEM, DL                  ; 代码段界限已在定义时设置好
        MOV     PSEG.BASEH, DH
        MOV     AX, DSEGD.BASEL                 ; 初始化目标数据段描述符
        MUL     BX
        ADD     AX, OFFSET BUFFER               ; 加上 BUFFER 的偏移地址
        ADC     DX, 0
        MOV     DSEGD.BASEL, AX
        MOV     DSEGD.BASEM, DL
        MOV     DSEGD.BASEH, DH
        LGDT    QWORD PTR VGDTR                 ; 加载 GDTR
        CLI                                   ; 关中断（未建立 IDT，程序必须运行在关中断下）
        ENABLE_A20                              ; 打开 A20 地址线
        TO_PROTECT                              ; 切换到保护模式
        JUMP16  <PCODE_SEL>, <OFFSET PROTECT>   ; 清指令预取队列，进入保护模式
REAL:   DISABLE_A20                             ; 关闭 A20 地址线
        MOV     AX, DATA                        ; 恢复 DS
        MOV     DS,AX
        STI                                     ; 开中断
        …                                       ; 根据具体任务要求实现“显示字符”程序段
CODE    ENDS
PCODE   SEGMENT USE16                           ; 定义 16 位代码段
        ASSUME  CS:PCODE
PROTECT:                                        ; 现在开始在保护模式下运行
        MOV     AX, DSEGS_SEL                   ; 加载源数据段描述符
        MOV     DS, AX
        MOV     AX, DSEGD_SEL                   ; 加载目标数据段描述符
        MOV     ES, AX
        CLD
        XOR     SI, SI                          ; 设置指针初值
        XOR     DI, DI
        MOV     CX, 256/4                       ; 设置以 4 字节为单位的数据长度
        REPZ    MOVSD                           ; 传送
        TO_REAL                                 ; 将 CR0 寄存器中的 D0 清 0，切换回实模式
        JUMP16  <SEG REAL>, <OFFSET REAL>       ; 清指令预取队列，进入实模式
PCODE   ENDS
        END     START
```

说明：

① 在从实模式切换到保护模式前，基本的准备工作是建立合适的全局描述符表 GDT，并使 GDTR 指向 GDT。因为在切换到保护模式时，至少把保护模式下代码段的选择子装载到 CS，所以 GDT 中至少含有保护模式下代码段的描述符。

電子工業出版社

本例全局描述符表 GDT 有 4 个描述符：第 1 个是空描述符，可以保证 GDT 中的第 1 个描述符永远不会被访问；第 2 个是代码段描述符……各描述符中的段界限是在定义时设置的，除了伪描述符 VGDTR 中的界限按 GDT 的实际长度设置，其余各段描述符的界限都规定为 0FFFFH。另外，描述符中的段属性也根据所描述段的类型被预置，92H 表示可读/可写数据段，98H 表示只可执行代码段，各段的特权级都设为 0。

② 由于在切换到保护模式后就要引用 GDT，因此在切换到保护模式前使用指令 LGDT 装载 GDTR，其功能是将 48 位伪描述符 VGDTR 装入全局描述符表寄存器 GDTR 中。低字是以字节为单位的全局描述符表段的界限，高双字为描述符表段的线性基地址（本例不启用分页机制，所以线性地址等同于物理地址）。

③ 在做好准备后执行 TO_PROTECT 宏指令，CPU 转入保护模式。安排一条段间转移宏指令：JUMP16　PCode_Sel, <OFFSET PROTECT>，在实模式下被预取并在保护模式下被执行。利用这条段间转移指令可把保护模式下代码段的选择子装入 CS，同时刷新指令预取队列。从此真正进入保护模式。

④ 在保护模式下实现数据传送时，需将源数据段和目标数据段的选择子装入 DS 和 ES，这两个描述符已在实模式下设置好，将选择子装入段寄存器意味着把包括基地址在内的段信息装入到段描述符高速缓冲寄存器。然后设置 SI、DI、CX 的初值，利用串操作指令实施传送。回到实模式下的显示是在常规内存中操作，代码很容易理解，故省略未写。

实际上，保护模式编程还可能涉及 16 位段和 32 位段的切换、多任务之间的切换以及中断描述符表 IDT、局部描述符表 LDT、任务状态段 TSS 的使用等。在此不再详述。

4.6.5 浮点指令程序设计

采用浮点指令的汇编语言程序格式，与整数指令源程序格式类似，仅需注意：① 编程时需使用 FPU 选择伪指令.387 等；② 使用伪指令 DD、DQ、DT（MASM 6.11 建议采用 REAL4、REAL8、REAL10）定义 32/64/80 位的浮点数据；③ 程序的第一条指令必须是初始化 FPU 的指令 FINIT，这表示开始执行一个新的浮点指令程序。

【例 4-28】 用浮点运算指令构建一个 0°～90°的高精度正弦表。

```
.MODEL SMALL
        .386
        .387                                ; 汇编 80387 指令
        .STACK
        .DATA
FCONST  DQ      180.0
TEMP    DW      0
FSINE   DQ      91 DUP(?)                   ; 共 91 个正弦函数值
        .CODE
        .STARTUP
        MOV     SI, OFFSET FSINE
        MOV     CX, 91
START1: FINIT                               ; 初始化 FPU
        FLDPI                               ; 压入 π
        FLD     FCONST                      ;压入 180
        FDIV                                ; 计算 π÷180
```


電子工業出版社

```
START2:  FILD     WORD PTR TEMP            ; 压入 0～90
         FMUL     ST(0), ST(1)             ; 度转换成弧度
         FSIN                              ; 求正弦函数值
         FSTP     QWORD PTR [SI]           ; 保存
         INC      WORD PTR TEMP            ; 计算下一度
         ADD      SI, 8
         LOOP     START2
START3:  .EXIT    0
         END
```

4.7 汇编语言与 C/C++语言混合编程

将汇编语言与 C/C++高级语言结合起来混合编程，可以充分利用系统与硬件技术的支持，取长补短，是编写高质量程序，特别是编写实时性要求比较高的程序的有效方法。

使用汇编语言与 C/C++语言混合编程的方法通常有两种：内嵌模块或多模块编程。

4.7.1 内嵌模块方法

内嵌模块方法是在 C/C++程序中直接嵌入汇编语言指令的语句。这种方法简单，只需在 C/C++程序中的_asm{}模块中嵌入汇编语言程序段即可。对于 Visual C++，除了在_asm{}模块中嵌入汇编语言程序段，还可以在汇编指令的前面用保留字“_asm”进行标记。

【例 4-29】 用汇编语言与 C/C++语言混合编程，实现在屏幕上显示字符串。

分析：可用汇编语言指令通过 EDX 寄存器传递待输出字符串的地址，然后调用 C 语言的标准输出函数 printf，将字符串输出显示在屏幕上，这里使用了嵌入汇编语言的两种格式。

```
#include<stdio.h>
char const *MESSAGE ="The planet Earth.\n$";
char *output ;
void main(void)
{
    _asm    MOV   EDX, MESSAGE          或   _asm {  MOV   EDX, MESSAGE
    _asm    MOV   output, EDX                        MOV   output, EDX   }
    printf("MESSAGE IS:%s", output);   //在屏幕上显示字符串
}
```

注意，混合编程时，能够使用的伪指令较少，如 OFFSET 这样的伪指令就不能用。

4.7.2 多模块混合编程

混合编程时，还可以采用之前介绍的模块化编程方法。将不同语言的程序分别编写并作为一个独立过程保存（如汇编语言程序），用 PUBLIC 伪指令将过程声明为公共的，提供给其他程序（如 C/C++语言程序）调用。C/C++语言程序在调用时，使用 EXTERN 伪指令声明所调用的过程（或函数）为外部的，同时指明该过程（或函数）的类型。

为了保证能够正确链接，各模块必须遵守相同的约定规则，这里以 C/C++语言程序对汇编语言程序的调用为例进行说明。

電子工業出版社

1．存储模式与标识符命名规则

通常，C/C++语言程序与汇编语言程序需具有相同的存储模式，以便自动产生相互兼容的调用与返回类型。

汇编语言程序中与 C/C++语言程序共享的变量名、过程名等标识符名称，都必须在前面加上下画线。如果在存储模式选择语句.model 中，增加了参数“c”，如.model　MEDIUM, c，则表明此汇编语言模块是按 C/C++语言规范编写的，这是 C/C++语言程序默认的调用约定，其入口参数和返回值均为 32 位。此时，不必在共享名称前加下画线。

2．寄存器使用规则

汇编语言程序可以任意使用通用寄存器(E)AX、(E)BX、(E)CX、(E)DX 和 ES，其中的(E)AX、(E)DX 承担了传递返回值的任务。对于段寄存器 DS、CS、SS 和(E)BP、(E)SP，汇编语言程序在使用时必须先保护（入栈），退出前再恢复。对于(E)SI、(E)DI，仅当 C/C++语言程序未启用寄存器变量时，汇编语言程序可直接使用，否则也必须先保护再使用，退出前恢复。

3．过程调用的参数传递规则

在 C/C++语言程序中，被调用的过程（函数）可以带参数，对它的说明形式如下：

```
EXTERN　返回值类型　函数名称(参数列表)
```

其中，返回值类型是 C/C++语言中函数允许的任意类型，当返回值类型省略时默认为 int 型，参数在传递过程中要求个数、类型和顺序一一对应。

所有高级语言在调用过程（函数）时，都是通过堆栈来传递参数的。C/C++语言程序从汇编语言程序中获取参数时，实模式程序规定用 AX、DX，保护模式程序规定用 EAX、EDX。而将参数从 C/C++语言程序传递给所调用的汇编语言程序时，必须遵守如下规则：

① 将 C/C++语言函数中的参数从右向左压入堆栈。例如，在 C/C++语言中函数 sum(x, y, z)的数据传输顺序是 z、y、x，即先将 z 压入栈区，然后是 y、x。C/C++语言传递普通参数时，传递的是参数本身；传递数组时，传递的是数组的指针。

② 参数压栈后，将当前的 CS、IP 压栈保护。如果 C/C++语言程序是在 TINY、SMALL 或 COMPACT 存储模式下编译的，则汇编语言程序中应将过程设置成 NEAR 型，将只保存 IP。如果 C/C++语言程序是在 MEDIUM、LARGE 或 HUGE 模式下编译的（过程被设置成 FAR 型），则先将 CS 压栈再将 IP 压栈。

注意使用的 C/C++语言程序开发环境。如果是 Visual C++ 6.0 环境，则只能编写 32 位应用程序，汇编子模块程序中必须用如.386P 这样的伪指令来说明采用的指令集，并选用 FLAT 存储模式。32 位 Visual C++中没有近、远调用之分，所有调用都是 32 位的偏移地址。同时不能使用 DOS 和 BIOS 功能调用指令（INT　n），因为这些功能调用只适用于 16 位应用程序。

③ 汇编语言程序的最后一条指令应该是不带参数的 RET。因为汇编语言程序返回后，断点自动弹出，然后 C/C++语言程序重新获得控制权。

【例 4-30】 在 Visual C++ 6.0 环境下编写一个函数，将三个数值传递给汇编过程，汇编过程对三个数求和后再将其传回 C++程序，由 C++程序将这个和在屏幕上显示。

设计汇编过程是 NEAR 类型，在调用汇编过程时，C++语言程序通过堆栈将三个数值

電子工業出版社

传递，同时将 EIP 压栈保护（32 位编程环境的寄存器是 32 位的）。汇编程序用 EBP 寻址堆栈，得到参数的偏移地址应为 EBP+8、EBP+12、EBP+16，堆栈的变化如图 4-12 所示。

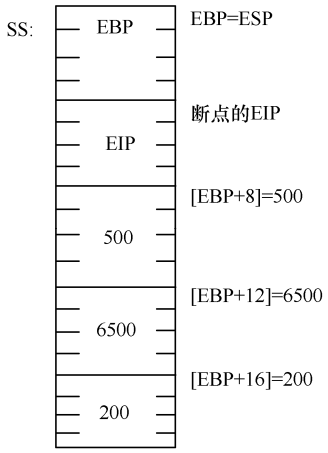


图 4-12　堆栈变化

因为数据类型被定义成 int 型，汇编过程通过 EAX 将计算出的和传递给 C++程序。参考程序如下（在 VC 6.0 和 MASM 32 V10.0 版本编译调试通过）：

```
        #include<stdio.h>    // following is main.cpp
        extern int sum (int, int, int);
        main()
        {
           printf("begin count\n");
           printf("The sum is equal to %d", sum(500, 6500, 200));
        }
        .386p                // following is sum.asm
        .MODEL   FLAT, C
        .CODE
        PUBLIC   SUM
SUM     PROC     NEAR
        PUSH     EBP
        MOV      EBP, ESP
        SUB      EAX, EAX
        MOV      EDX, EAX
        ADD      EAX, [EBP+08H]
        ADC      EDX, 0
        ADD      EAX, [EBP+0CH]
        ADC      EDX, 0
        ADD      EAX, [EBP+10H]
        ADC      EDX, 0
        POP      EBP
        RET
SUM     ENDP
        END
```


電子工業出版社

习 题 4

4-1　汇编语言程序的开发流程是怎样的？分别利用什么程序完成？产生什么输出文件？

4-2　区分下列概念，并说明标号和变量的属性：

（1）标号和变量　（2）数值表达式和地址表达式　（3）符号常量和字符串常量

4-3　给出下列语句中立即数（数值表达式）的值：

（1）MOV　AX, 1234H/16 + 10H

（2）MOV　AX, 23H SHL 4

（3）MOV　AL, 'H' OR 00100000B

（4）MOV　AX, (76543 LT 32768) XOR 7654H

4-4　以图示说明下列语句分配的存储空间及初始化数值：

```
VAR1    DB   'ABC', 10, 10H, 'EF', 2 DUP(-1, ?, 3 DUP (4))
VAR2    DW   10H, -5, 'EF', 3 DUP (?)
VAR3    DW   VAR2+4
VAR4    DD   VAR2
```

4-5　设置一个数据段 DATASEG，按照如下要求定义变量：

（1）MY1B 为字符串变量：PERSONAL COMPUTER

（2）MY2B 为用十进制数表示的字节变量：20

（3）MY3B 为用十六进制数表示的字节变量：20

（4）MY4B 为用二进制数表示的字节变量：20

（5）MY5W 为 20 个未赋值的字变量

（6）MY6C 为值是 100 的常量

4-6　对于下列数据定义，在括号内写出各指令语句独立执行后的结果：

```
NUM_B      DB     2 DUP (?)
NUM_W      DW     10 DUP (42H)
ARRAY_B    DB     'DISP_IMAGE'
```

（1）MOV　BX，TYPE　NUM_W　　; BX=（　　）

（2）MOV　AL，LENGTH　NUM_B　　; AL=（　　）

（3）MOV　CX，SIZE　ARRAY_B　　; CX=（　　）

（4）MOV　DL，LENGTH　NUM_W　　; DL=（　　）

4-7　设某数据段定义如下：

```
D_SEG    SEGMENT  PARA 'DATA'
ORG      30H
DATA1    EQU   $+20H
VAR1     DB    10 DUP (?)
VAR2     DW    'AB', 2, 2000H
CNT      EQU   $-VAR1
VAR3     DB    'HELLO'
D_SEG    ENDS
```

试回答：

（1）变量 VAR1、VAR2、VAR3 的偏移地址各是多少？

（2）常量 DATA1、CNT 的值分别是多少？

電子工業出版社

（3）VAR2+2 单元的内容为多少？

4-8　指出下列每对伪指令语句的区别：

（1）X　DB　89H　　; Y　EQU　89H

（2）X　DW　7805H　　; Y　EQU　BYTE PTR X

（3）X　EQU　38　　; Y = 38

（4）X　DW　6288H　　; Y　DB　62H, 88H

（5）X　DW　5816H　　; Y　DW　5816

4-9　假设 MYWORD 是一个字变量，MYBYTE1 和 MYBYTE2 是字节变量，指出下列语句中的错误原因。

（1）MOV　BYTE PTR [BX], 1000

（2）MOV　AL, MYBYTE1+ MYBYTE2

（3）CMP　MYBYTE1, MYBYTE2

（4）SUB　AL, MYWORD

（5）JNZ　MYWORD

4-10　设在某个程序中有如下片段，请写出每条传送指令执行后寄存器 AX 的内容。

```
DATA    SEGMENT
        ORG     100H
VARW    DW      1234H, 5678H
VARD    DD      12345678H
BUFF    DB      10  DUP(?)
MESS    DB      'HELLO'
        …
BEGIN:  MOV     AX, OFFSET MESS
        MOV     AX, TYPE BUFF+TYPE MESS+ TYPE VARD
        MOV     AX, SIZE VARW+SIZE BUFF+ SIZE MESS
        MOV     AX, LENGTH VARW + LENGTH VARD
        MOV     AX, LENGTH BUFF + SIZE VARW
        MOV     AX, TYPE BEGIN
        MOV     AX, OFFSET BEGIN
```

4-11　写出具有下列功能的伪指令语句（序列）。

（1）将字数据 2786H、23H、1A24H 存放在定义为字节变量 DATA1 的存储区中，并且不改变数据按字存储时的次序。

（2）将字节数据 30H、0B4H、62H、10H 存放在定义为字变量 DATA2 的存储区中，并且不改变数据按字节存储时的次序。

4-12　按下面要求写一个完整段定义格式的源程序。

（1）数据段从字边界开始，其中定义一个 100 字节的数组，同时该段还作为附加段；

（2）堆栈段从节边界开始，组合类型为 stack；

（3）代码段的类别是 code，指定段寄存器对应的逻辑段；给有关段寄存器赋初值；将数组元素全部设置为 64H；按标准序返回 DOS。

4-13　按下面要求写一个简化段定义格式的源程序。

（1）定义常量 num，其值为 5；数据段中定义字数组变量 datalist，它的头 5 个字单元中依次存放-1，0，2，5，4，最后 1 个单元初值不定。

電子工業出版社

（2）代码段中的程序将 datalist 中头 num 个数的累加和存入 datalist 的最后 1 个字单元中。

4-14　代码段中开始的一段程序有通用性，试将此段定义为一条宏指令。

4-15　定义一条宏指令，实现任一数据块的传送（假设无地址重叠），只要给出源和目标数据块的首地址以及数据块的长度即可。

4-16　设 x、y、z 已定义为字节变量。若 x 和 y 各存放一个 32 位（4 字节）的无符号数，存放顺序是低位字节存入低地址，试写出将 x 和 y 相加、结果存入 z 的程序段。

4-17　将内存中某一区域的数据块传送到另一区域中。考虑它们的地址可能重叠：先判断源首址加数据块长度是否小于目标首址，若小于，表明源数据块与目标数据块的地址没有重叠，按增量方式传送；反之，则调整地址指针按减量方式传送。定义相应数据区并完成程序设计。

4-18　某程序共可接收 10 个键盘命令（分别为 A，B，C，…，J），完成这 10 个命令的程序分别为过程 P0，P1，…，P9。编写程序从键盘接收命令，并转到相应的过程去执行。要求用两种方法：（1）用比较/转移指令实现；（2）用跳转表实现。

4-19　试编写一程序段，完成两个以压缩 BCD 格式表示的 16 位十进制数（8 字节）的加法运算，相加的两个数 x 和 y 以及和 SUM 可定义为字节变量，并假定高位存入低地址。

4-20　从 FIRST 开始的 100 个单元中存放着一个结束符为“$”的字符串。编写一个程序，统计该字符串中字母 A 的个数。

4-21　试编制一程序，统计 DATA_W 字数组中正数、负数和 0 的个数，并分别保存在变量 COUT1、COUT2、COUT3 中。

4-22　试找出无符号字节数组 ARRAY 中的最大偶数与最小偶数，最大偶数送 MAX 单元，最小偶数送 MIN 单元。

4-23　编写一个程序，将变量 ABC 中的 16 位无符号数用“连续除以 10 取余”的方法转换成十进制数，转换结果用压缩 BCD 码保存在 RESULT 开始的单元中。

4-24　有一最大长度为 80 个字符的字符串已定义为字节变量 STRING，试编写程序，找出第一个空格的位置（位置以 00H～4FH 表示）存入 CL 中，若该串无空格符，以-1 存入 CL 中。

4-25　对题 4-24，若该字符串以回车符结束，试编写程序，统计该串的实际长度（不包括回车符），统计结果存入 CH 中。

4-26　在数据段中有字符串的定义如下，使用串操作指令编写完成下列功能的程序段（设 DS 和 ES 重叠）：

```
STRING1     DB  'MESSAGE AND PROCCESS'
STRING2     DB  20 DUP (?)
```

（1）从左到右把 STRING1 中字符串搬到 STRING2 中；

（2）从右到左把 STRING1 中字符串搬到 STRING2 中；

（3）搜索 STRING1 字符串中是否有空格，如有，记下第一个空格的地址，放入 BX 中；

（4）比较 STRING1 和 STRING2 字符串是否相同。

4-27　内存缓冲区存有一个以“$”结尾的字符串 STRING。编制子程序，根据入口参数 AL=0、1、2，分别实现大写字母转换成小写、小写字母转换成大写或大小写互换。

4-28　编制一个子程序把一个 16 位二进制数用十六进制形式在屏幕上显示出来，分别用如下三种参数传递方法，并用一个主程序验证它。

（1）采用 AX 寄存器传递这个 16 位二进制数。

（2）采用 WORDTEMP 变量传递这个 16 位二进制数。

電子工業出版社

（3）采用堆栈方法传递这个 16 位二进制数。

4-29　用模块化编程方法实现字符串"ASSEMBLER"的显示，要求模块 1 为主程序，负责数据初始化工作；模块 2 中定义子程序 DISPLAY 显示字符串；主程序与子程序 DISPLAY 在不同的代码段中。

4-30　编写一个计算校验和的子程序。所谓“校验和”，是指不计进位的累加，常用于检查信息的正确性。主程序提供入口参数，包括数据个数和数据缓冲区的首地址。子程序则返回求和结果。

4-31　编程实现：从键盘输入 4 位十六进制数的 ASCII 值，将其转换成 4 位十六进制数存入 DX 寄存器中。

4-32　编写程序，输入 10 个一位十进制数，将它们从大到小排序后在屏幕上显示出来。

4-33　假设 100 名学生的某门课程成绩存放在数据区中，试编写程序，统计：成绩低于 60 分的人数，60～90 分的人数，高于 90 分的人数，并显示在屏幕上。

4-34　读取系统日期并显示。

4-35　用条件控制伪指令和循环控制伪指令分别实现的两个程序段如下：

（1）
```
.IF     (X == 5) && (AX != BX)
        INC     AX
.ENDIF
```

（2）
```
.WHILE      AX != 10
        MOV    [BX], AX
        INC     BX
        INC     BX
        INC     AX
.ENDW
```

要求用汇编指令重新设计上述分支、循环结构程序，并比较汇编程序生成的代码序列。

4-36　使用.REPEAT 和.UNTIL 语句编写一子程序，查找是从 DS:SI 开始的内存单元中与 AL 相同的数值，设查找区域以 0 结束。若找不到，CF=0，否则 CF=1，且 SI 指向所找到位置。

4-37　计算 32 位有符号数组的平均值（求和时考虑溢出），结果置于 EAX，余数存入 EDX。

4-38　上机实现例 4-27 中 8 个元素（每个元素 16 位，1 个字）的向量点积计算程序。

4-39　编程产生从 1～10 整数的平方根表，结果以单精度浮点数存在 ROOT 表中。

4-40　用混合编程方法实现冒泡排序程序。将排序算法写成一个汇编模块，然后编写一个 C/C++语言主程序，提供待排序的数据和显示排序后的结果。

電子工業出版社

第 5 章　半导体存储器

5.1　半导体存储器概述

存储器是计算机实现记忆功能的核心部件，用于存放待处理的数据和中间计算结果以及系统或用户程序等供处理器读写和执行。显然，容量越大（即能存储的信息越多）、存取速度越快、成本越低，则性能价格比越高。但是，这三方面的性能之间常常存在矛盾，较好的解决方法是采取分级存储结构，通过合理的设计获得快慢搭配、协调工作的存储系统。典型的现代微机系统的存储组织如图 5-1 所示，呈现金字塔形结构，越往上存储器件的存取速度越快，CPU 的访问频度越高，同时单位存储量的价格也越高，占系统的存储容量的比例越小。可以看到，CPU 片内寄存器位于塔顶，存取速度最快，但数量很少；向下依次是 CPU 内的 Cache（高速缓冲存储器）、主板的 Cache、主存储器、辅助存储器（半导体盘、磁盘等，一般以联机方式工作）和大容量辅助存储器（光盘、磁带等，可脱机工作）。位于塔底的存储设备的容量最大，单位存储成本最低，但存取速度可能较慢或最慢。

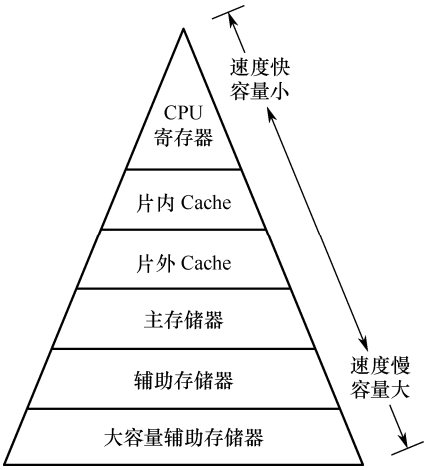


图 5-1　微机系统分层存储结构

简单地从微处理器角度看，存储器分为两类：一类是可以通过标准数据总线连接直接访问的内部存储器，用于存储当前与 CPU 频繁交换的信息，其工作速度快，但容量较小；另一类是外部存储器，用于存储 CPU 暂不处理的信息，其容量很大，故也称为海量存储器，即所谓的外存。外存中的信息既可被方便地修改，又可长期保存，但 CPU 需要配置专门的接口和驱动设备才能实现访问，对它的存取速度也较内存慢得多。当其信息需要处理时，要先通过接口电路调入内存，再由 CPU 处理。高速缓存通过专门的控制电路，提供大容量内存的高效、高可靠的分级访问功能。

在内存方面，随着大规模集成电路技术的发展，半导体存储器的集成度、可靠性和存取速度等性能迅速提高，制造工艺更为简便，而成本迅速下降。在外存方面，采用快擦写存储器（Flash ROM）的 U 盘目前已基本取代了传统的软磁盘，成为主要的移动存储手段，而低功耗、高可靠的固态硬盘也开始逐步代替机械式大容量磁介质硬盘。

为了便于系统扩展和升级，部分微处理器引入了专门的存储器管理单元（MMU），通过分段、分页及保护机制，支持高速虚拟存储器，实现多用户、多任务应用系统。

本章先介绍存储器的主要指标，典型半导体存储器的组成、结构、工作原理以及存储器的容量扩充方法，然后介绍 8086/8088 CPU 的存储器系统设计，讨论 80386/Pentium CPU 系统的存储器连接以及多级存储器和虚拟存储器技术。

電子工業出版社

5.1.1 半导体存储器的分类

从制造工艺的角度分类，半导体存储器可分为双极型和金属氧化物半导体型。

① 双极（Bipolar）型半导体型由 TTL（Transistor-Transistor Logic）晶体管逻辑电路构成。这类存储器件的工作速度快，可与 CPU 处在同一量级，但集成度低、功耗大、价格偏高，在现代微机系统中常用作高速缓存器、控制存储器。

② 金属氧化物半导体（Metal-Oxide-Semiconductor）型简称 MOS 型，有多种制作工艺，如 NMOS、HMOS、CMOS、CHMOS 等，可用来制作多种半导体存储器件，如静态 RAM、动态 RAM、EPROM 等。这类存储器的集成度高、功耗低、价格便宜，但存取速度较双极型器件慢。微机的内存主要为 MOS 型。

按应用与存取方式不同，半导体存储器又分为随机存取存储器 RAM（Random Access Memory）和只读存储器 ROM（Read Only Memory）两大类，如图 5-2 所示。

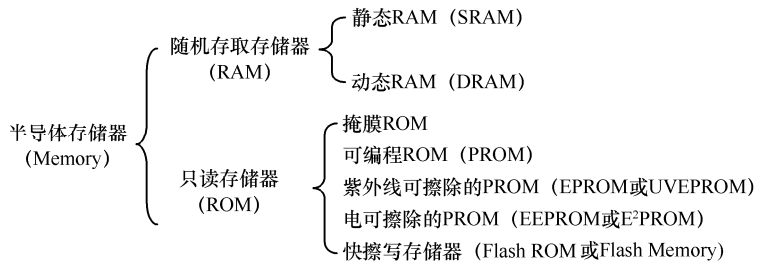


图 5-2 半导体存储器分类

① RAM 在使用中可由程序随时读/写存储单元内容，一般用于存放输入/输出数据、中间结果或加载的用户应用程序。

② ROM 在工作时，其内容只能读出，不能写入，一般用于存放工作程序或固定不变的参数表等。掩膜 ROM 的内容是不能改变的，而 EPROM 等的内容在特定条件下（如紫外线光照或高电压、大电流、特殊时序等）才能擦除改写。这类存储器存储的内容在断电后仍能长期保持，具有非挥发性（Non-Volatile），其中 E^2PROM，Flash ROM 可支持电擦除，并可由程序在线改写。此外，还有一类结合了两者特性的非挥发随机存取存储器（NVRAM）。

微机工作时，一般先由 ROM 中的引导程序启动系统，需要时再从外存中读取系统程序和应用程序，加载到内存 RAM 中；在多数嵌入式系统应用中，往往直接运行存放在只读存储器中的系统和应用程序。在程序的运行过程中，中间结果一般存放在 RAM 中，程序也可根据需要将结果送入外存。保存在外存中的程序和数据可以根据需要被调入内存再次运行或修改。

近年来，为满足微处理器应用的需要，各种大容量、高速、高集成度、低功耗存储器及相关技术发展很快，先后出现了伪静态存储器 PSRAM、高速缓存器 Cache、快速页模式 FPM-RAM、扩展数据输出 EDO-RAM、Rambus-DRAM 和各种同步高速 RAM（同步动态 SDRAM、同步图形 SGRAM、双倍及多速率 SDRAM，即 DDR、DDR2、DDR3 等）、高性能大容量快擦除存储器等技术。大容量 RAM 在使用上则将多片甚至多种存储芯片组合集成到一起，配上控制电路构成标准存储器模块（Memory Module，习惯上又称为内存条）。

電子工業出版社

5.1.2 存储原理与地址译码

用于微机系统的半导体存储器芯片结构一般如图 5-3 所示，由保存数据的存储体矩阵、片内地址译码电路、译码驱动电路、读/写控制电路、三态缓冲器和控制逻辑等组成。

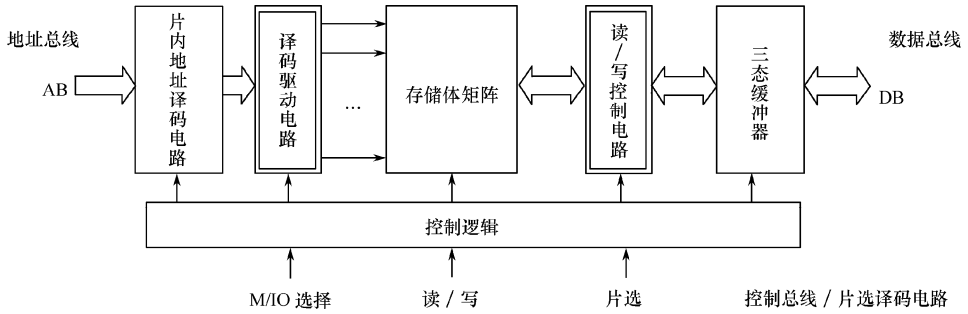


图 5-3 半导体存储器芯片的结构

1. 存储体

一个基本存储电路（或称为记忆单元）能存储 1 位二进制数，若干基本存储电路组成一个存储单元，如一个 8 位的二进制数需 8 个基本存储电路。容量为 $M\times N$（如 64K×8）位的存储器则包含 $M\times N$ 个基本存储电路。这些基本存储电路有规则地排列在一起便构成了存储体，是存储器的基础和核心。面向微机的存储单元一般可组合存储 1 字节（8 位）或 1 字节的 2、4、8 倍信息，即存放 8、16、32、64 位二进制信息。不同类型存储器具有不同的存储体结构，以实现快速读/写、大容量存储或非易失存储等性能。

2. 地址译码与驱动

译码与驱动电路一般包含译码器和驱动器两部分。译码器（Decoder）是将每个编码信号译成一个特定输出信号的电路，存储器一般产生 N 选 1 的线性有序输出，即对于某一地址编码信号，N 个输出线上有唯一的有效（高或低）电平信号与之对应。例如，5 位二进制编码信号可得到 32 个译码信号，当 $A_4A_3A_2A_1A_0$= (00000)时，输出译码信号 X_0～X_{31} 中仅有 X_0 输出为有效信号（如高电平），其他 X_1～X_{31} 输出均为无效信号（如低电平）。$A_4A_3A_2A_1A_0$= 00001 时，仅有 X_1 有效，其他均为无效，以此类推。

为了区分存储体中具体的存储单元，给每个单元规定一个编号，标识相应的存储单元，简称地址。为了准确读/写指定的存储单元，计算机需要采用专门的地址译码电路，包括片内译码和片选译码两级：片选译码采用高位地址，片内译码利用低位地址信号进行译码。

常用的片内地址译码有单译码和双译码两种方式。

（1）单译码方式

图 5-4 是一个 N 选 1 译码器。译码器输出驱动 $N=2^n$ 条译码信号线，若单元 I/O 字线由 M 位组成，某条字线被选中，则对应此线上的 M 位信号便同时被读出或写入，经输出缓冲放大器输出或输入一个 M 位的字。在图 5-4 中，若 n=4，字线 N=16，M 为 8 位，则地址译码器的地址输入线 n 应为 4 位，2^4=16 个状态，分别控制 16 条字线（S_0～S_{15}）。当地址信号为 0000 时，选中字线 S_0，若进行读出操作，则该字线上的 8 位同时被读出；类推地址信号 0001 对应 S_1，0010 对应 S_2……地址信号 1111 选中第 16 条字线 S_{15}。此时如果是写操作，则该字线上的 8 位便同时被写入存储单元。

電子工業出版社

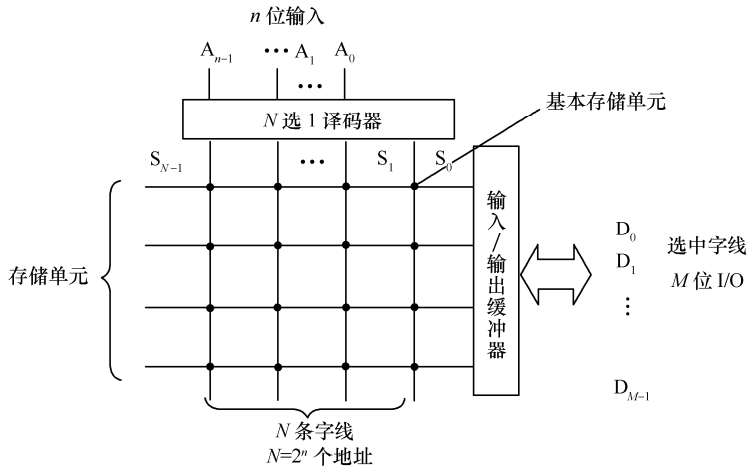


图 5-4　存储器单译码方式

单译码方式主要用于早期的小容量半导体存储器，对于现代大容量的存储器，则多采用 X/Y 双译码方式。

（2）X/Y 双译码方式

X/Y 双译码方式又称为复合译码方式，在内部采用两级译码电路。由于地址译码信号个数 n 较大时，字单元选择线的条数 N 必然很多，这时可将 n 个信号分成 q 和 r 两组，如 $N=2^n=2^{q+r}=2^q\times 2^r=X\times Y$，对 N 个状态的译码分别由 X 译码和 Y 译码两部分完成。

以 n=12 为例，可以分为 $N=2^{12}=2^6\times 2^6=64\times 64=4096$，其双译码方式如图 5-5 所示。

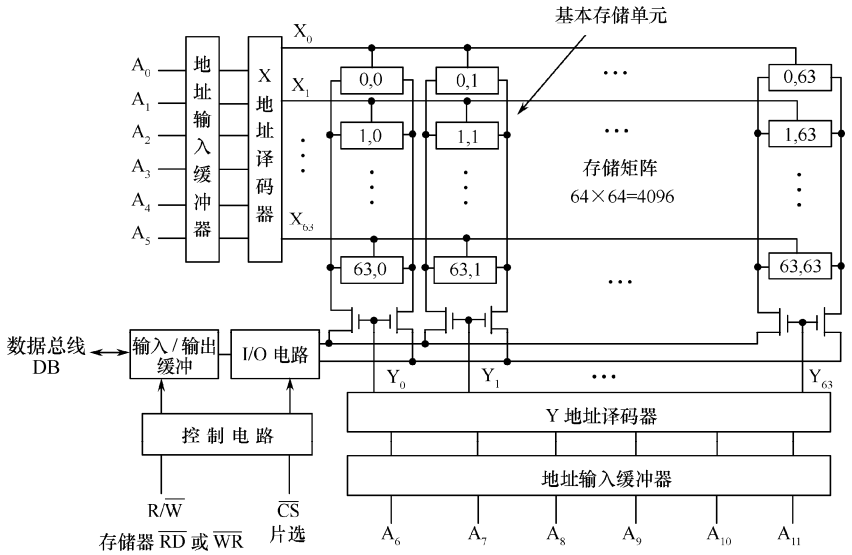


图 5-5　存储器双译码方式

4096 个单元排成 64×64 的矩阵，需要 12 条地址线 A_{11}～A_0，分成 X 和 Y 两部分，A_5～A_0 输入至 X 方向（行）译码器，它输出 64 条选择线，分别选择 64 行（X_0～X_{63}）；A_{11}～A_6 输入至 Y 译码器，输出的 64 条位选择线分别选择 Y_0～Y_{63} 列，控制各列的位线。设 A_{11}～A_0=000000000000，则 X 地址译码输出 X_0 为高电平有效，选中 X_0 行，则 X_0 行控制的(0, 0)，(0, 1)，(0, 2)，…，(0, 63)各位都可被选中进行读（写）操作，具体选中哪一位，取决于 Y

地址译码，在该图例中，Y 地址译码输出 Y_0 应为高电平，列线 Y_0 有效，此列控制的位线控制门打开，故 X、Y 双向译码结果是选中(0, 0)对应一组基本存储电路，即可对这一组存储电路进行读（写）操作。如果一个存储字为 8 位，就需要 8 个这样的存储单元。当一个地址被选中时，构成该地址单元的 8 位阵列状态数据同时被读出（或写入）。

在半导体存储器中，一个或几个 X、Y 阵列被集成在一块芯片上。如果将存储字的 8 位都集成在一块芯片内，这种存储器为字结构芯片，如 Intel 2764（EPROM）、Intel 6116（RAM）存储容量分别为 8K×8 位、2K×8 位；如果芯片中集成的是各存储字的同一位或几位，这种芯片则称为位结构芯片，如 Intel 2164A（RAM）、μPD424256（RAM）存储容量分别为 64K×1 位和 256K×4 位。

与单译码方式比较，双译码寻址可显著减少输出选择线的数目。单译码时 K 位地址需要 2^K 个状态输出线，在双译码结构中，假设 K 为偶数，则 $K/2$ 个输入端可以有 $2^{K/2}$ 个输出状态，X 和 Y 两个地址译码器共有 $2^{K/2}\times 2^{K/2}=2^K$ 个输出状态，译码输出线却只有 $2^{K/2}+2^{K/2}$ 条。以 16 位地址为例，若采用单译码方式，译码输出需要 2^{16}（即 65536）条选择线；若采用 X、Y 各 8 线双译码方式，构成 256×256 的矩阵，输出状态仍为 65536 个，但译码输出选择线只需 256+256=512 条，大大减少了选择线的数目。存储器容量越大，这种存储结构的优点越突出。

3．控制逻辑

读/写控制电路包括读出放大器、写入电路和读/写控制电路，以完成对被选中单元中各位的读出或写入操作。控制逻辑接收来自片选译码（$\overline{\text{CS}}$ 或 $\overline{\text{CE}}$）、存储器读/写信号，经读/写控制电路实现存储单元的读/写操作。

微机系统中存储器的读/写操作是在 CPU 的控制下对指定单元进行访问，只有当接收到来自 CPU 的存储器读/写命令后，才能实现正确的读/写操作。三态缓冲器提供存储单元与数据 I/O 总线的正确操作，当未访问该存储器芯片时，三态缓冲器输出处于高阻（Z）状态。

5.1.3 主要性能指标

（1）存储容量

存储器的容量是指一个存储器芯片所能存储的二进制信息量。

存储器容量=存储单元数×I/O 数据线位数（字长）

如某芯片有 2048 个存储单元，每个单元存放 8 位二进制数，则其容量为 2048×8（16384）位，或写为 2048×8 位（16384）。为简化书写，习惯上以 1K 表示 1024（故容量 2048×8 位简记为 2K×8 b 或 16K b）。同样，以 1M 表示 1K×1K 等。

目前，CPU 的字长已经达到 16、32、64 位，可以访问 1、2、4、8 字节单元。在微机系统中，存储器几乎都以字节为基本单元，所以常用存储器的字节数来表示系统存储器的容量，如 8088（PC/XT）存储空间 1M，80386 的 4M 存储器，均指字节数（1 MB 和 4 MB）。

（2）最大存取时间

存储器的存取时间定义为一次访问存储器（对指定单元写入或读出）所需时间，这个时间的上限值即最大存取时间，一般为几纳秒到几百纳秒（ns）。最大存取时间越短，芯片的工作速度越快。

電子工業出版社

（3）供电电压、逻辑电平、接口方式

传统的存储器多采用+5 V 供电，标准 TTL 逻辑电平，并行数据接口。近年来，低电压 CMOS 技术大量应用，+3.3 V、+2.5 V、+1.8 V、+1.5 V（分别称为 LVTTL、SSTL_2、SSTL_18、SSTL_15 兼容）等标准电压和逻辑电平的陆续引进，为实现低功耗、高速运行提供了条件；另外，采用串行接口可以大大简化存储器接口电路，提高可靠性，因此串行接口被广泛用于 E^2PROM 甚至高速 RAM 芯片接口。串行接口技术详见第 7 章。

另外，面向多处理器或控制器接口的存储器，如同步和异步的 FIFO-RAM、双口（Dual Port）RAM 或串行双口 E^2PROM 也成为设计中的一类重要选择。

（4）其他指标

功耗、可靠性、集成度、价格等也是用户所关心的指标。

需要说明的是，实际选型时上述各项指标往往相互矛盾，要求存储器同时具有这些高性能参数是困难的，应根据实际情况权衡，并根据需要合理设计存储器结构。

5.2 随机存取存储器（RAM）

5.2.1 静态 RAM（SRAM）

1. SRAM 原理

（1）基本存储电路

基本存储电路主要由 R-S 触发器构成，其两个稳态分别表示存储内容为 0 或 1。容易理解，SRAM 在正常电源供电时，存入的数据可以保存和读出；只要不写入新的数据，触发器状态就保持不变，读（触发器的输出）操作不会改变它；但 RAM 芯片在断电后，原存储的信息将全部丢失，这就是所谓的“易失性”。

（2）外围电路

与一般存储器类似，外围电路包括如下。

① 地址译码器：对输入的外部地址信号译码，用以选择要访问的单元。在目前所用的 SRAM 存储器中，通常采用双译码结构，如图 5-6 所示。

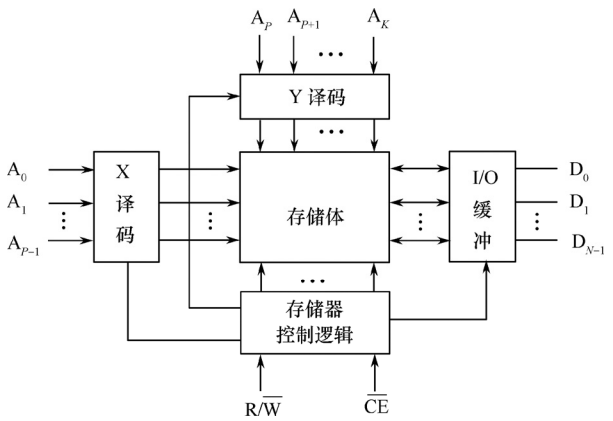


图 5-6 SRAM 基本结构

② I/O 控制电路：处于数据总线和被选中的单元之间，如用 R / $\overline{W}$（读/写控制信号，

電子工業出版社

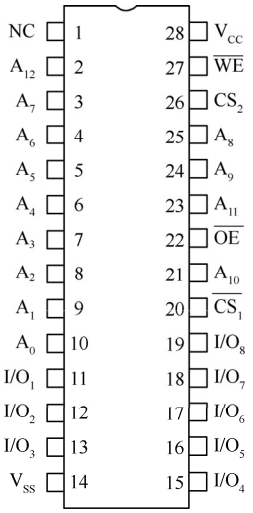
也可用独立的存储器读（$\overline{RD}$）和存储器写（$\overline{WR}$），负脉冲有效）来控制被选中单元的读出或写入。$\overline{CE}$（或$\overline{CS}$）为片选信号，当$\overline{CE}$为低电平时，该芯片被选中使能，并能正常读/写信息；而当$\overline{CE}$为高电平时，表示该芯片未被选中。

2．典型存储器芯片 HM6264BL

以低功耗 CMOS SRAM HM6264BL 为例，其容量为 8K×8 b，采用单+5 V 供电，输入/输出电平与 TTL 兼容，最大存取时间为 70～120 ns。

（1）HM6264BL 引脚

HM6264BL 芯片采用双列直插封装，28 个引脚，引脚及引脚信号含义如图 5-7 所示。



(a) 引脚图

引　脚	功　能
A_0~A_{12}	地址输入
$\overline{WE}$	写信号
$\overline{CS_1}$,CS_2	片选
$\overline{OE}$	输出允许
I/O_1~I/O_8	数据线
V_{CC}	电源
V_{SS}	地
N.C	未用

(b) 引脚含义

图 5-7　SRAM HM6264BL

（2）HM6264BL 的工作方式

表 5-1 为 HM6264BL 工作方式功能表(真值表)。当$\overline{CS_1}=1$或$CS_2=0$时芯片不被选中。当$\overline{CS_1}=0$、CS_2=1、$\overline{OE}=0$、$\overline{WE}=1$时，为存储器读方式；当$\overline{CS_1}=0$、CS_2=1、$\overline{WE}=0$时，为存储器写方式。HM6264BL 采用典型的高/低电平（CS_2和$\overline{CS_1}$）双片选控制，主要是因为在早期电池供电的存储器系统设计时，需要构造特殊的掉电保护电路，当供电电压低于一定值时封锁CS_2。

表 5-1　HM6264BL 工作方式功能表

$\overline{CS_1}$	CS_2	$\overline{WE}$	$\overline{OE}$	工作方式	I/O 信号
1	×	×	×	低功耗	高阻
×	0	×	×	低功耗	高阻
0	1	1	1	输出禁止	高阻
0	1	1	0	读	数据输出
0	1	0	×	写	数据输入

（3）HM6264BL 读/写周期时序（以读取时间 70 ns 的 HM6264BL 为例）

存储器的存取时间是存储器的重要指标，HM6264BL 读/写周期和典型工作时间参数如下。

① 存储器的读周期。图 5-8 为 HM6264BL 读周期典型时序，其中主要参数说明见表 5-2。根据图 5-8，实现存储器读操作必须使$\overline{CS_1}$和$\overline{OE}$为低，CS_2为高。当给出的地址信号

電子工業出版社

经过一段地址访问时间（读存储器）t_{AA}，读出的数据就会出现在外部数据线上，这段时间称为读取时间。HM6264BL 最大只需 70 ns。但是，数据能否送到外部数据总线上，还取决于片选和输出允许信号，从 $\overline{CS_1}$、CS_2 和 $\overline{OE}$ 有效到内部总线稳定时间分别为 t_{CO} 和 t_{OE}。

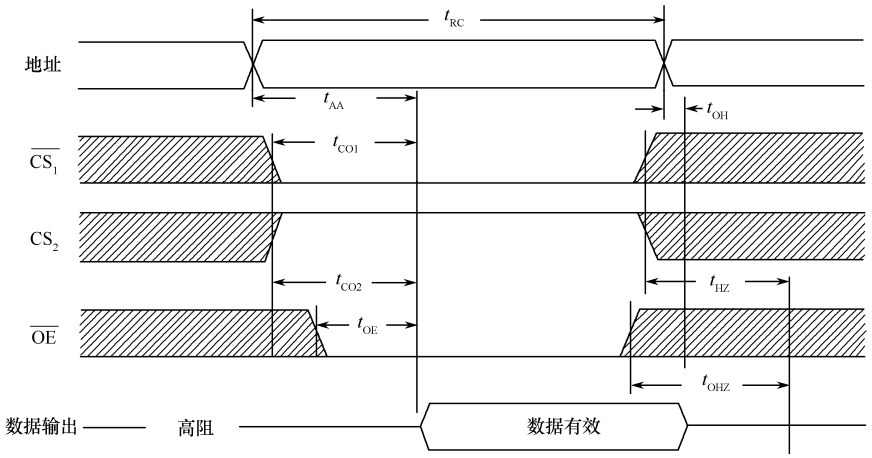


图 5-8　存储器 HM6264BL 读周期时序

表 5-2　HM6264BL 读周期主要参数说明

参　数	符　号	时间（ns）	
		最　小	最　大
读周期时间	t_{RC}	70	—
读出时间	t_{AA}	—	70
片选到输出	t_{CO}	—	70
输出允许到输出稳定	t_{OE}	—	35
输出禁止到数据线高阻	t_{OHZ}	10	—
片选无效到数据线高阻	t_{HZ}	0	30
地址改变后数据保持	t_{OH}	10	—

即对于存储器读，需从地址信号有效起，经过 t_{AA} 时间以后进行，而且从片选有效和输出允许有效起经过 t_{CO} 和 t_{OE} 后，数据才能稳定输出，这两者必须同时保证。

读周期和读取时间是两个不同的概念。读周期表示该芯片进行两次连续的读操作时起点必须间隔的最小时间，故它总是大于或等于读取时间。

② 存储器写周期。图 5-9 为 HM6264BL 写周期典型时序，主要参数说明如表 5-3 所示。

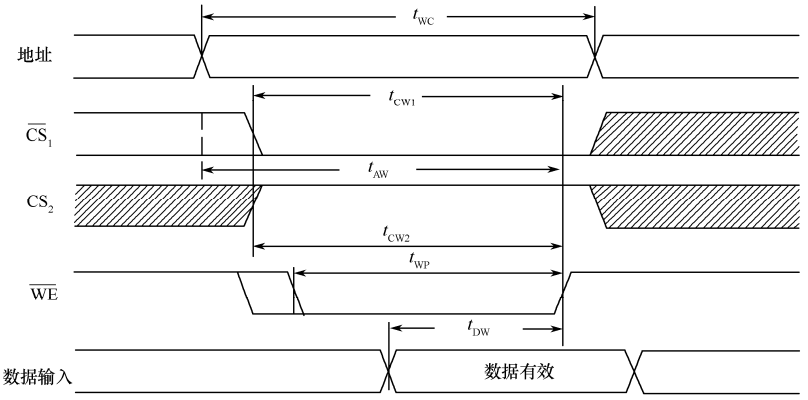


图 5-9　存储器 HM6264BL 写周期时序

電子工業出版社

表 5-3　HM6264BL 写周期主要参数说明

参　数	符　号	时间（ns）	
		最　小	最　大
写周期时间	t_{WC}	70	
片选宽度	t_{CW}	60	
地址有效到写结束	t_{AW}	60	
写脉冲宽度	t_{WP}	40	
数据保持到写有效	t_{DW}	30	

根据图 5-9，实现写操作必须 $\overline{CS_1}$、CS_2、$\overline{WE}$ 都有效。注意，在地址改变期间，$\overline{WE}$ 必须为高，避免误写入操作。

5.2.2　动态 RAM（DRAM）

1．单管基本动态存储电路

基本动态存储电路是利用 MOS 管栅极和源极之间的电容 C 来存储信息的，图 5-10 为单管基本动态存储电路。以电容上“有”“无”电荷两种状态来区分其存储的二进制信息是“1”还是“0”。由于工艺问题，一般 C 的电容值小于数据线的分布电容 C_D，每个数据读出后，C 上的电荷经 C_D 释放，信息易被破坏，所以每个数据读出后，要重新恢复 C 上的电荷量。即使无“读操作”，电容随着时间变化放电，电荷泄漏也会造成信息丢失。为保持 C 中信息（电荷），需周期性地充电，这一过程称为（动态）刷新。刷新周期通常为 2～8 ms。

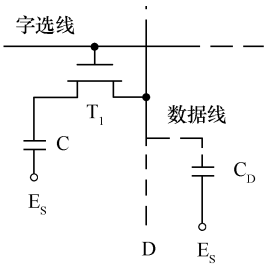


图 5-10　单管 DRAM 基本动态存储电路

与 SRAM 相比，DRAM 的基本存储电路较简单，故集成度较高、功耗小，除了正常读/写，额外增加了刷新操作，需要相应的外部电路支持，设计使用较复杂。另外，DRAM 的存取速度一般比 SRAM 慢。

DRAM 一般用于组成大容量 RAM 存储器，而容量不大时（如常规小型系统或高速缓存）选用 SRAM 芯片更为简单实用。

2．典型芯片 μPD424256

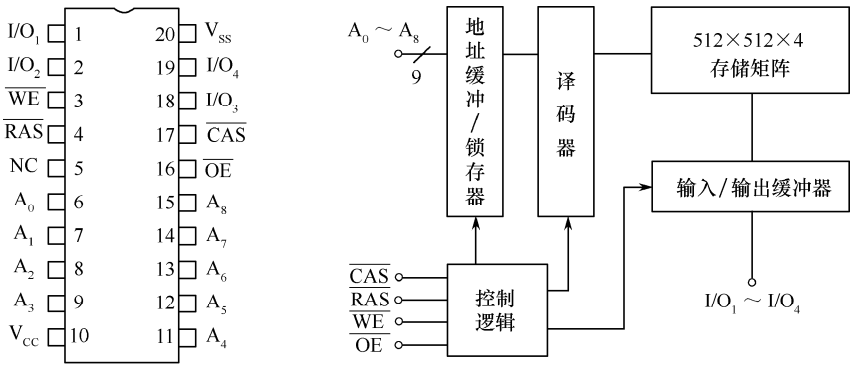
（1）内部结构

μPD424256 芯片的容量是 256K×4 b，片内译码需 $\log_2 256K=18$ 条地址线。为了减少封装引线、缩小体积，地址线被分成两部分：行地址和列地址，这样芯片的外部地址引线只有 9 条，行、列地址分时送入，利用内部多路开关和锁存器，由行地址选通信号 $\overline{RAS}$ 把先出现在地址线上的 9 位地址信号锁存至行地址锁存器，然后列地址选通信号 $\overline{CAS}$ 把后出现在地址线上的 9 位地址锁入列地址锁存器，如图 5-11 所示。

（2）读/写控制

行列地址译码后选中 256K 个单元中的一个单元进行访问，读/写由 $\overline{OE}$ 和 $\overline{WE}$ 控制。当 $\overline{WE}$ 有效（低电平）时，表示写入，I/O 线上的信号经输入三态缓冲器写入被选中单元；当 $\overline{WE}$ 无效（高电平）时，表示读出，$\overline{OE}$ 有效时被选中单元的内容经输出三态门出现在 I/O 线上。μPD424256 无片选信号 $\overline{CS}$。$\overline{RAS}$ 和 $\overline{CAS}$ 均有效即表示芯片被选中。

電子工業出版社



(a) 引脚图　　(b) 内部结构图

图 5-11　μPD424256 芯片引脚及内部结构

（3）μPD424256 工作时序

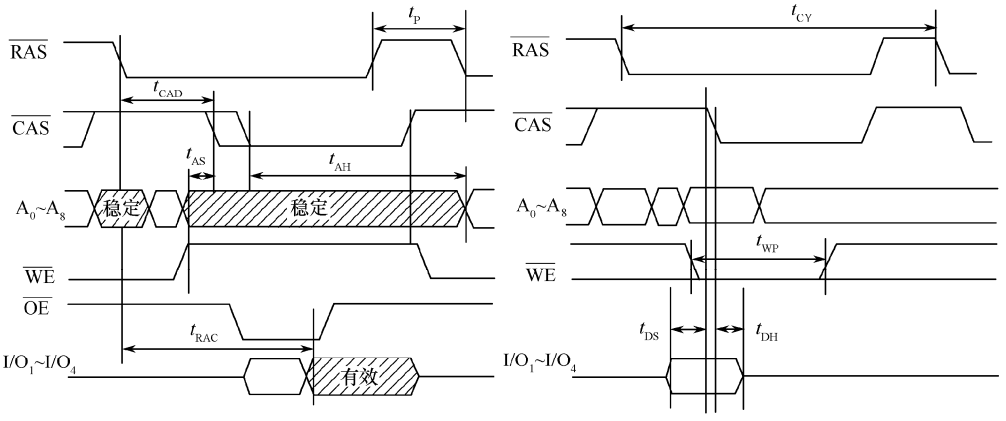
① 读周期（时序如图 5-12(a)所示）。为了使芯片能正确地接收行、列地址并实现读操作，各信号间的关系应满足如下条件：

- 行地址必须在 $\overline{\text{RAS}}$ 信号有效之前送到芯片的地址输入端，以确保被选中的单元信息正确读出。
- $\overline{\text{CAS}}$ 信号应迟后 $\overline{\text{RAS}}$ 一段时间，并迟后于列地址送到芯片地址输入端的时间。
- $\overline{\text{RAS}}$、$\overline{\text{CAS}}$ 应有一定宽度，才能保证被选中单元信息的正确读出。
- 读有效（$\overline{\text{WE}}=1$）应在 $\overline{\text{CAS}}$ 有效之前建立。

② 写周期（如图 5-12(b)所示）。写周期中 $\overline{\text{RAS}}$ 与 $\overline{\text{CAS}}$ 之间的关系，以及它们与地址信号之间的关系和读周期相同。不同的是：

- 写命令（$\overline{\text{WE}}=0$）应在 $\overline{\text{CAS}}$ 信号有效之前建立，且在 $\overline{\text{RAS}}$（或 $\overline{\text{CAS}}$）变高前结束。
- 写入的数据必须在 $\overline{\text{CAS}}$ 有效之前出现在 I/O 引脚，才能保证数据可靠地写入。

③“读-修改-写”时序。在指令中，常常需要对某个单元的内容读出后进行修改，再写回该单元。为提高操作速度，存储器设计了专门的“读-修改-写”周期，基本上是读/写周期的组合。



(a) 读周期　　(b) 写周期

图 5-12　μPD424256 芯片读/写周期

④ 页模式周期。为提高数据块操作时数据传送的速度，存储器还设计有页模式操作。

電子工業出版社

在该模式下，$\overline{RAS}$ 不变，由连接的 $\overline{CAS}$ 信号对不同的列地址进行操作。注意，这时 $\overline{RAS}$ 脉冲宽度有一个最大限度，即每次能连续扫描的列数是有限的。在页模式操作时，可实现读/写及“读-修改-写”操作。

（4）刷新周期

刷新操作是通过执行只有 $\overline{RAS}$ 信号的 DRAM 访问周期来实现的。刷新时，在 $\overline{RAS}$ 信号期间行地址有效，选中存储矩阵中的一行单元，将其内容分别读至相应的内部读出放大器后又重新写回单元。对由行地址决定的 512 个组合逐一施加 $\overline{RAS}$ 信号，μPD424256 芯片内部的所有单元都进行一次刷新，刷新周期为 8 ms。注意，因在刷新期间 $\overline{CAS}=1$，刷新数据不会出现在 I/O 线上。

5.2.3 随机存取存储器 RAM 的应用

在实际应用中，随着 CPU 存取速度的不断提高，要求存储器的读/写速度随之提高。在技术上，一方面要提高存储器容量和速度，主要通过 DRAM 改进，缩短延迟、提高带宽并结合应用采取专门措施帮助提高系统性能；另一方面需要引入高速缓存技术，提高系统级存储器读/写速度。

1. 基于预测技术的 DRAM

要加快普通 DRAM 的访问速度，简单实用的方法是在芯片上附加一些逻辑电路。以 DRAM 为例，这些附加电路包括地址多路转换电路、地址选通、刷新逻辑以及读/写控制逻辑等。通过增加少量的附加逻辑电路，可以提高单位时间内的数据流量，即通过预测增加带宽的技术。较为成功的技术包括快速页模式 FPM（Fast Page Mode）和扩展数据输出 EDO（Extended Data Output），相应的存储器称为 FPM-DRAM 和 EDO-DRAM。

① 快速页模式 FPM。通常在 DRAM 阵列中读取一个单元时，首先提供一个行地址，并置 $\overline{RAS}$ 为低电平，然后通过周期性的列信号 $\overline{CAS}$ 实现多个列存取。$\overline{RAS}$/$\overline{CAS}$ 选择时充电线路在稳定前会有一定的延时，制约了存储器的读/写速度。在大多数情况下，要存取的数据在 RAM 中是连续存放的，即下一个要存取的单元常常位于当前单元的下一地址，FPM 采用这一预测技术（这里的页指 DRAM 芯片一行存储单元中的一个 2048 位区间），增加了快速页读或写操作来缩短页模式周期。

② EDO 模式。在 FPM 的基础上，增加了超页（Hyper Page）读或写以及超页“读-修改-写”等操作，即利用地址预测，可在当前读/写周期中启动下一个存取单元的读/写周期，进而在宏观上缩短地址选择的时间。采用这一技术，可比 FPM 提高效率 10%～15%。

此外，在 EDO 技术上引入突发模式，即假定对若干后续地址进行预取操作。

2. 同步 DRAM（SDRAM）

上述 FPM-RAM 和 EDO-RAM 都属于异步内存，其输入/输出过程不与 CPU 同步，CPU 在存取数据时必须保持和监测各种信号，等待 DRAM 完成所有的内部操作。典型的等待时间为 50～70 ns，这个延迟时间也就代表了异步 DRAM 的存取速度。

相比之下，SDRAM（Synchronized DRAM）则具有同步接口。它的基本原理是，将 CPU 和 RAM 通过一个相同的时钟锁在一起，使它们共享一个时钟周期，以相同的速度同步工作。即在系统时钟的控制下，SDRAM 能够把地址、控制和数据信号同步起来，经过指定的时钟

電子工業出版社

数后，CPU 可以直接从数据总线上获得所需数据或将数据写到总线存入相应的存储单元。由于不再需要监测数据的存取过程，一些具有乱序执行功能的高级 CPU 能够在等待存取数据时完成其他工作，这也是 SDRAM 的优点之一。SDRAM 芯片的速度通常用其最高输出频率来表示，如 66 MHz、100 MHz、133 MHz，转换成时间分别为 15 ns、10 ns、7.5 ns。但是当多片 SDRAM 被集成为模块（内存条）使用时，实现各种信号之间的同步则会引入额外的延迟，因此模块的稳定工作频率要低于芯片最高输出频率。

SDRAM 基于多存储体结构，内含 2 或 4 个交错的存储阵列体（bank），当 CPU 从一个阵列体访问数据的同时，另一个阵列体已准备好读/写数据。通过存储阵列体的紧密切换，读取效率可得到成倍的提高。

SDRAM 芯片还支持突发传输模式，在这种工作模式下，当第一个列地址输入之后，芯片内部自动产生后续若干（2、4、8 个或 FP 全页）连续的列地址，从而可以快速输出后续地址的数据（预取）。突发数据长度可通过修改突发计数器对应的寄存器来设定。SDRAM 与支持突发模式的高速缓存相配合，可以有效地提高系统的性能。“全页”方式时，只在时钟脉冲的上升沿对行地址选通信号 $\overline{\text{RAS}}$ 及其有关的控制信号进行采样，之后不再需要这些信号，也就是说，$\overline{\text{RAS}}$ 信号仅需持续一个时钟周期，而呈现为一个窄脉冲。这不同于传统的 DRAM，后者通常要求 $\overline{\text{RAS}}$ 信号在整个操作过程中始终保持有效。

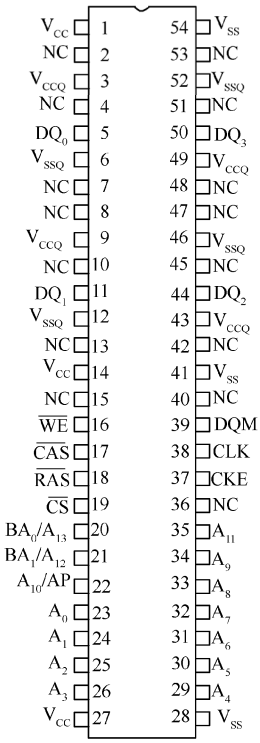
SDRAM 通常支持两种刷新方式：自动刷新（Auto-refresh）和自刷新（Self-refresh）。前者为标准方式，当 CKE 信号有效且时钟允许时进行刷新；后者当芯片处于低功耗情况（CKE 及时钟禁止）时芯片自行刷新。

LGS 公司的 GM72V28441 是典型的 SDRAM 芯片，其容量为 8M×4 b×4 bank，符合 PC66/100/133 标准，LVTTL 接口，3.3 V 电源，每 64 ms 包含 4096 个刷新周期，具有双刷新操作，可编程突发访问模式，顺序或间隔，长度 1、2、4、8 或 FP（全页）；可编程 $\overline{\text{CAS}}$ 延迟 2 或 3 个时钟；4 个存储阵列块可独立或同时操作；可突发读/突发写或者突发读/单独写。图 5-13 为 GM72V28441 芯片引脚图及功能框图。

在 SDRAM 技术基础上升级发展起来的 DDR SDRAM（Double Date Rate SDRAM）简称 DDR，也称为 DDR1，即双倍速率 SDRAM，采用双数据速率架构实现高速操作，目前其第四代产品（DDR4）已得到成熟广泛应用。DDR 在时钟信号上升沿与下降沿各传输一次数据，因此数据传输速率是传统 SDRAM 的 2 倍。由于仅多采用了下降沿信号，因此并不会造成能耗增加。至于地址与控制信号则与传统的单速率（SDR）SDRAM 相同，仅在时钟上升沿传输。在技术上，DDR SDRAM 工作电压 2.5 V，接口电平 SSTL_2，通过精准时序控制达到性能与成本之间的折中。DDR SDRAM 需要通过锁相环 PLL 和自标定保证时序精度，同时可通过工作频率的不断提高，弥补带宽上的不足。从最初的 DDR200 开始，PC266 DDR SRAM（133 MHz 时钟×2 倍数据传输=266 MHz 带宽）是由 PC133 SDRAM 内存衍生出来的，它促使 DDR 内存大量应用，随后出现 DDR333 到 DDR400、DDR533 内存，并出现双通道 DDR400 内存，成为 800 FSB（前端总线）处理器搭配的基本标准。

随着 CPU 性能不断提高，对内存性能的要求逐步升级。仅仅依靠高频率提升带宽的 DDR 毕竟能力有限，因此出现了 DDR2、DDR3、DDR4 标准和应用，预取缓冲深度达到 4、8、16（位），可以在一个时钟周期内传输 4 次、8 次、16 次数据，综合性能提高明显。DDR SDRAM 特性参数如表 5-4 所示，各标准之间不兼容；更高性能的 DDR5 等标准也正在酝酿发布中，并有望首先用于手机及移动终端（MDDR）和图形处理（GDDR5、GDDR6）。

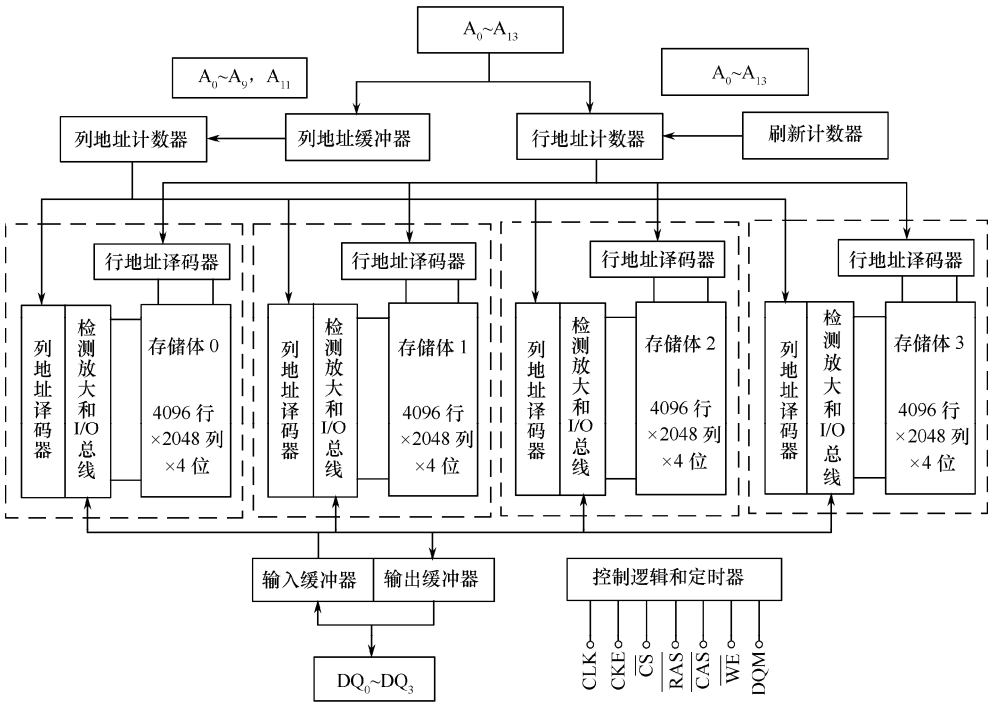
電子工業出版社



(a) 引脚图

引脚名	功　能
CLK	时钟
CKE	时钟允许
$\overline{CS}$	片选
$\overline{RAS}$	行地址选通
$\overline{CAS}$	列地址选通
$\overline{WE}$	写允许
A_0~A_9,A_{11}	地址输入
A_{10}/AP	地址输入/自动预充电
BA_0/A_{13}~BA_1/A_{12}	存储体选择
DQ_0~DQ_3	数据输入/输出
DQM	数据输入/输出屏蔽
V_{CCQ}	数据输入/输出电源
V_{SSQ}	数据输入/输出地
V_{CC}	内部电路电源
V_{SS}	内部电路地
NC	未用

(b) 引脚说明



(c) 功能框图

图 5-13　8M×4B×4 存储体 SDRAM 芯片 GM72V28441

電子工業出版社

表 5-4　DDR SDRAM 标准主要技术参数

DDR 标准	发表年份	总线时钟/MHz	内存时钟/MHz	预取（最小突发）	传输速率/(TM/s)	电压/V	DIMM 引脚数	SO-DIMM 引脚数	Micro-DIMM 引脚数
DDR1	2000	100～200	100～200	2n	200～400	2.5/2.6	188	200	172
DDR2	2003	200～533.33	100～266.67	4n	400～1066.67	1.8	240	200	214
DDR3	3007	400～1066.67	100～266.67	8n	800～2133.33	1.5/1.35	240	204	214
DDR4	2014	800～1600	200～400	8n	1600～3200	1.2/1.05	288	256	—

3．基于协议/内部串行接口的 RDRAM

RDRAM（Rambus DRAM）是 DDR SDRAM 相对昂贵的替换品，由美国 Rambus 公司开发。与 DDR 和 SDRAM 不同，RDRAM 引入了新一代高速简单内存架构，采用串行数据传输模式，使整个系统性能得到提高。从本质上讲，RDRAM 是一种基于协议的 DRAM。由于种种原因，如无法保证与原有的制造工艺兼容、专利费用、制造成本及价格高昂等而被淘汰。

5.3　只读存储器（ROM）

ROM 存储的信息是非易失的，也就是说，断电后再通电，存储内容不会改变。一般通过编程写入，应用时，其内容只能读出、不能改写。这两点是各类 ROM 存储器的共性，也是与 RAM 的本质差异。但简单地看，ROM 的基本结构与 SRAM（见图 5-6）基本相同。

5.3.1　掩膜 ROM 和 PROM

掩膜 ROM 芯片存储的信息是由芯片制造厂商完成的，用户不能修改，图 5-14 是一个典型的 2×2 位 ROM 存储矩阵。矩阵中，字选线与数据线之间或有晶体管 T 相连（如“字选线 0”与“D_0”），或无管子相连（如“字选线 0”与“D_1”），这些晶体管是制造时由二次光刻板的图形（掩膜工艺）决定的，故称为掩膜 ROM。图 5-14 中，T_4、T_3 是负载管，当字选线 0 选通（“1”）时，开关管 T_1 导通，则数据线 D_0 上输出为“0”，而 D_1 与字选通 0 不连通，D_1 上输出为“1”。可见掩膜 ROM 是由有/无晶体管来区分 0/1 信息的。

典型的 PROM（Programmable ROM，可编程 ROM）基本存储电路如图 5-15 所示。芯片出厂时，开关管 T_1 与位线之间以熔丝相连。用户可根据需要写入信息（熔断或保留熔丝以区分“1”或“0”），但只能写入一次（熔断后不能再连通）。PROM 的写入要由专用电路（大电流、高电压）和程序来完成。为了与 RAM 的随机写入过程区别，称 PROM 的写入过程为编程。

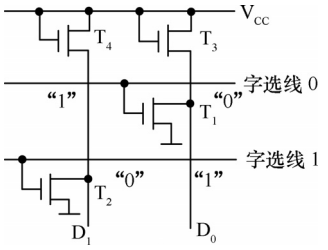


图 5-14　2×2 位掩膜 ROM 原理

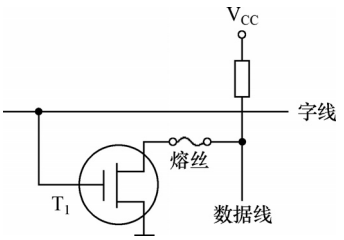


图 5-15　PROM 基本存储电路原理

電子工業出版社

5.3.2 EPROM（可擦除的 PROM）

根据片内信息擦除方式的不同，可擦除的 PROM 分为两类：① 紫外线可擦除的，即 UVEPROM（Erasable Programmable ROM，EPROM）；② 电可擦除的，即 EEPROM 或 E^2PROM（Electrically Erasable Programmable ROM）。

1．EPROM

（1）基本存储电路

EPROM 典型电路如图 5-16 所示。EPROM 与普通的 P 沟道增强型 MOS 电路相似，只不过源（S）和漏极（D）之间有一个由多晶硅制成的浮栅，被绝缘层 SiO_2 包围着。EPROM 芯片制造时，硅栅上无电荷，EPROM 管内无导电沟道。当用管子构建存储矩阵时，矩阵内容全为 1。编程写入时，在 D、S 间加上具有规定电压（如+12.5 V 或更高）、宽度约 50 ms 的编程脉冲，所选中的单元在编程电源的作用下，D、S 之间被瞬时击穿，就会有电子通过绝缘层注入硅栅。当高压电源去除后，因硅栅被绝缘层包围，故注入的电子无处泄漏，硅栅为负电位，于是就形成了导电沟道，从而使 EPROM 单元导通（字线与数据线相连，相当于逻辑 0）；而未被选中单元的 D、S 间未被击穿，字线与数据线不连通，仍为逻辑 1。

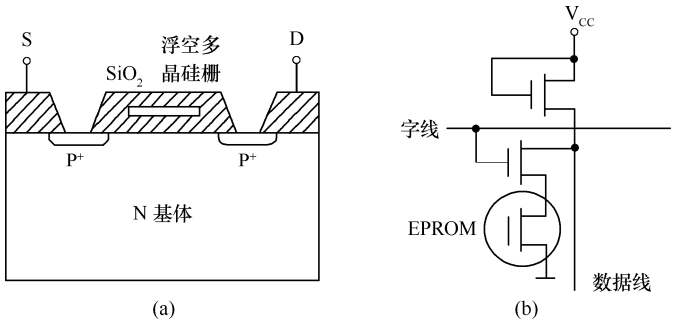


图 5-16　EPROM 基本存储电路原理

这种 EPROM 存储电路芯片外壳上方有透明的玻璃窗口，当用紫外线通过窗口照射时，所有电路的浮栅上的电荷形成光电流泄漏掉，从而把写入信号擦去（内容全恢复成 1），以便重新编程。这一过程通常由专用设备（EPROM 擦除器）完成，再用 EPROM 写入器实现编程。

（2）典型芯片 Intel 27C512

Intel 27C512 是容量 64K×8 b 的 CMOS EPROM 芯片，封装形式为 DIP 或 PLCC，引脚排列如图 5-17 所示，地址线为 $\log_2 64K=16$ 条，8 位数据线带三态输出缓冲。其读周期类似 SRAM，当引脚 $\overline{CE}$ 和 $\overline{OE}$ 都为低电平时数据输出。

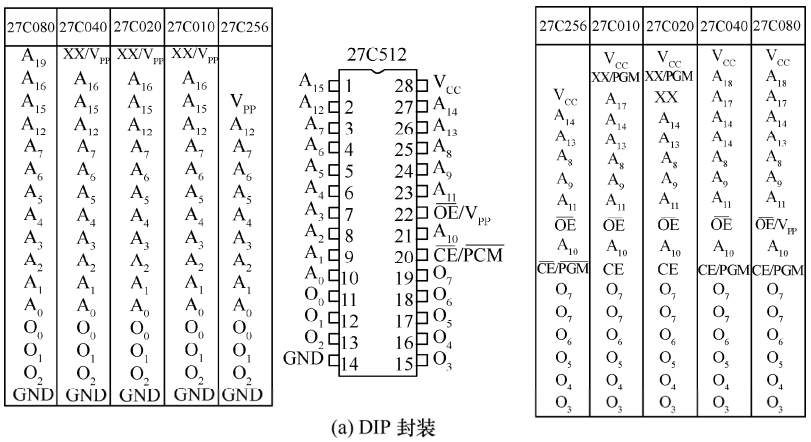
常用的 EPROM 芯片容量为 16K×8 b（27C128）至 1M×8 b（27C080），引脚有一定的兼容性，如图 5-17 所示，单片大容量可达 8 Mb（27C160，2M×8 b 或 1M×16 b）。

2．E^2PROM

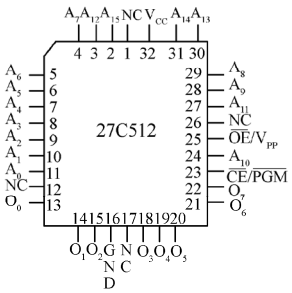
电擦除可编程只读存储器 EEPROM（或 E^2PROM）的主要特点如下：

① 编程简单，对硬件电路没有特殊要求。早期的 EEPROM 芯片是靠片外加高压电源（约+20 V）进行擦写的，如 2817 芯片，后来又把升压电路集成在片内，使得擦写在+5 V 电源下即可完成，如 2817A、2864 芯片等。

電子工業出版社



(a) DIP 封装



(b) PLCC 封装

引脚名	功　能
A_0~A_{15}	地址线
$\overline{CE}/\overline{PGM}$	片选 / 编程
$\overline{OE}$	输出允许
O_0~O_7	数据输出线
NC	未用

(c) 引脚含义

图 5-17　64K×8 位 EPROM 27C512 引脚信号及芯片引脚含义

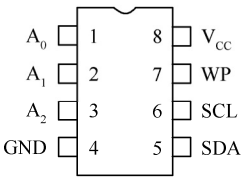
② 采用+5 V 电擦除的 E^2PROM，通常不需设置单独的擦除操作，可在写入过程中自动擦除。但擦除时间较长，约 10 ms，故要保证足够的写入时间。

③ E^2PROM 器件一般采用并行总线，容量不大，价格高于 EPROM，可与处理器总线直接连接。也有采用串行数据传送的，其容量可很小，安全可靠、价格低、体积小。

④ 通常需要设计专门的防写保护电路，以防止在上电或掉电过程中出现误写操作。

常用的并行 E^2PROM 芯片如 2864，其引脚与 EPROM 2764 完全兼容（与 27512 引脚兼容，可参考图 5-7 和图 5-17），最大存取时间 200 ns，编程和工作电压均为+5 V。

典型的串行 E^2PROM 如 AT24C01A，其容量为 128×8 b（1 Kb），引脚图及引脚含义如图 5-18 所示，片内地址、数据和命令等均由 SDA 线传送，其中 A_2～A_0 表明允许串行级联最多 8 片。串行 E^2PROM 外部连接简单，符合一定的串行总线标准（如 I^2C，详见第 7 章），同系列不同容量的存储器具有相同的外部引脚结构，如 AT24C02（2 Kb），AT24C04A（4 Kb），AT24C16A（1K×16 b）等，可达 1 Mb（AT24C1024，地址 A_1，最多 2 片）。



(a) 引脚图

引脚名	功　能
A_0~A_2	片地址
SDA	串行数据输入 / 输出
SCL	串行时钟输入
WP	写保护

(b) 引脚含义

图 5-18　串行 E^2PROM 存储器 AT24CxxA 引脚图及含义

電子工業出版社

3．OTP ROM（One Time PROM）

这是一种经济型的 PROM，除了没有擦除窗口，其他工艺与 EPROM 完全相同，但价格远远低于同容量的 EPROM。它可用普通的 EPROM 编程器对其编程，但只能编程一次。

典型芯片如 HN27C512、HN27C256 等，引脚与同容量的 EPROM 芯片完全相同。OTP 工艺也广泛用于带程序存储器的单片机（微控制器）、DSP 及嵌入式处理器芯片中。

4．快擦写存储器

快擦写存储器（Flash ROM 或 Flash Memory）又称为闪速存储器，简称闪存，是一种新型的可读/写、非易失半导体存储器，近年技术发展极快。与 E^2PROM 相比，Flash ROM 具有读/写速度快、容量大的特点；与 SRAM 相比具有非易失及价廉等优势，它突破了传统的存储器体系，改善了现有存储器的特性。从功能上讲，由于闪速存储器随机存取和非易失的 ROM 特点，因此 Flash ROM 能够用于程序代码和数据存储，对于需要实施代码或数据更新的嵌入式系统应用是一种理想的存储器。作为一种高密度、非易失的读/写半导体器件，Flash ROM 特别适合用于制作固态盘，并且优势明显。另外，其低成本和高可靠性也常用于替代电池支持的 SRAM。

（1）性能与分类

与 EPROM 不同，Flash ROM 可实现大规模电信号擦除，擦除整个存储器阵列或一个大的存储单元块。这种擦除过程较为复杂并需花费几秒钟的时间。擦除操作之后可以紧接着重新编程写入。这种编程写周期比 RAM 的写周期长得多。但它的擦除/重写次数是有限的，主要用于某些数据需要重写但又不经常改写的场合。

目前市场上主要的闪存多基于 NOR 和 NAND 两种结构。

① NOR 闪存技术开发于 1988 年，其特点是并行总线，易与处理器总线相连，实现芯片内执行（eXecute In Place，XIP），这样应用程序可以直接在闪存上运行，不必再把代码读到系统 RAM 中。实际上，NOR 闪存带有 SRAM 类似接口，有足够的地址引脚来寻址，可以很容易地存取其内部的存储字节。NOR 的传输效率很高，在 1 MB～4 MB 容量时具有很高的成本效益，但是很低的写入和擦除速度大大影响了它的性能。

② NAND 闪存技术始于 1989 年，更强调降低每位存储成本和更高的性能，并且像磁盘一样可以通过接口轻松升级。NAND 闪存使用复杂的 I/O 接口来串行传输存取信息，格式和方法可能各不相同，8 个引脚用来传送控制、地址和数据信息。NAND 结构能提供极高的单元密度，单元尺寸几乎是 NOR 闪存的一半，其生产过程更为简单，可以在给定的模具尺寸内提供更高的容量，相应地降低了价格。同时写入和擦除的速度很快，且读/写操作采用 512 B 块，易于磁盘文件的管理。因此，基于 NAND 的闪存很自然地被用于取代硬盘或其他块存储设备。事实上，几乎所有的 U 盘和 CF、SD、MMC 等存储卡都使用 NAND 闪存作为存储介质。应用 NAND 的困难在于，需要特殊的系统接口。除了接口，实际选用设计闪存系统时还需要解决介质的可靠性问题，包括寿命（耐用性）、位交换和坏块处理等。

在 NOR 器件上运行代码不需要任何软件支持，NAND 器件上进行同样读操作时，常需要相应的驱动程序。但 NAND 和 NOR 器件在进行写入和擦除操作时都需要驱动程序支持。

（2）Flash ROM 阵列结构

与 EPROM 和 E^2PROM 类似，Flash ROM 快速便捷的擦写是其重要的性能，有几种不同的阵列结构，这与为了达到擦除目的的电路组织方式有关。三种标准的 Flash ROM 阵列结构是整体擦除（Bulk Erase）、自举块（Boot Block）、快擦写文件（Flash File）。

整体擦除 Flash ROM 是比较基本的一类，表 5-5 列出了部分典型的整体擦除 Flash ROM

電子工業出版社

的型号、位密度及存储容量。

表 5-5　整体擦除 Flash ROM

Flash ROM 型号	位密度/b	存储容量/b
28F256	256K	32K×8
28F512	512K	64K×8
28F010	1M	128K×8
28F020	2M	256K×8
28F320	32M	4M×8 或 2M×16
28F640	64M	8M×8 或 4M×16
28F128	128M	16M×8 或 8M×16

自举块 Flash ROM 是为使用嵌入式微处理器应用而设计的。与整体擦除相比，其特点是：① 有更高的存储密度；② 其智能电压（Smart Voltage）能力允许用 5 V 或 12 V 的 V_{PP} 电压对该电路编程，也允许 V_{CC} 电压为 5 V 或 3.3 V；③ 通过某输入位选择字节宽度，能适应 8 位或 16 位总线；④ 硬件可封锁块。当自举块被封锁时，任何擦除或编程这个块的企图都将导致出错，这时称该自举块是写保护的；⑤ 支持自动擦除和写入操作，不需要用软件实现复杂的快速擦写编程算法。

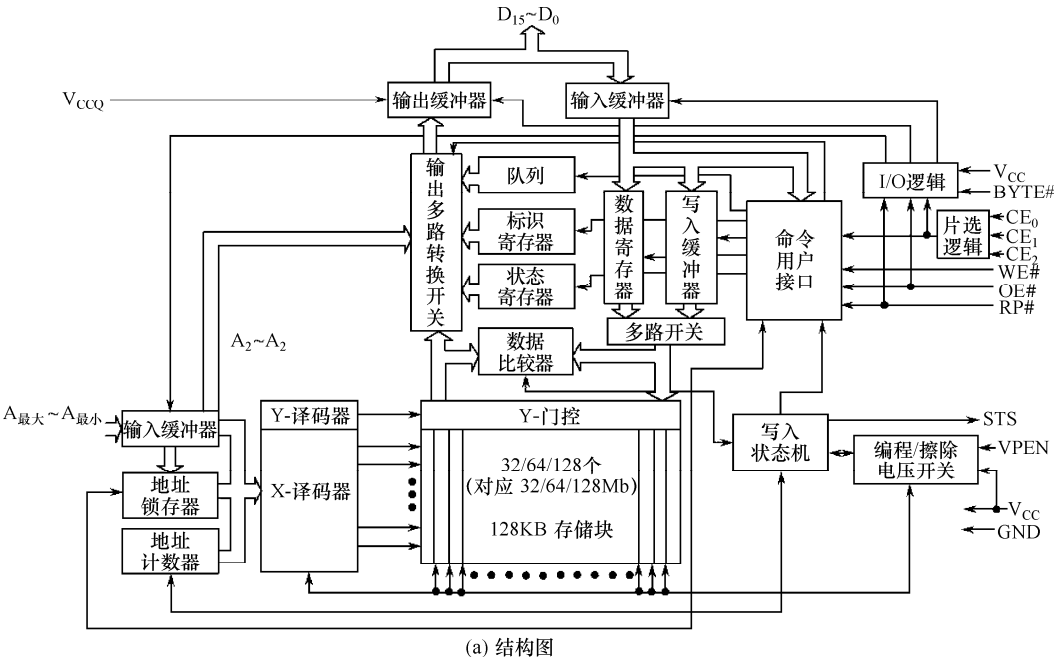
目前，高密度的 Flash ROM 多为 32、64、128、256 Mb 的电路，采用对称大小、独立可擦除的快擦写文件（Flash File）结构来将其存储阵列分块。这些电路适用于大容量的代码存储及固态信息存储电路（如快擦写卡及快擦写驱动器）的应用。与自举块结构相比，快擦写文件结构增加了一个输出信号，进一步减小在擦除和编程过程中对 CPU 软件的依赖性；也支持块封锁机制，但在这类电路中，在封锁和解除封锁时，每一块是独立可编程的。

（3）典型的 NOR 闪存 Intel Strata Flash

Intel Strata Flash 是 Intel 公司的 NOR Flash ROM 产品，典型芯片有 28F320J3/640J3/128，容量 32、64、128 MB，内部逻辑框图如图 5-19 所示，采取 0.18/0.25 μm 工艺，对称存储块，每块 128 KB，每块至少可擦写 10 万次。工作电源+3 V，访问速度有 110 ns、115 ns、120 ns、150 ns 共 4 档，25 ns 同步页模式读，32 字节写缓存，每字节有效编程时间 6.8 μs，具备 128 位加密寄存器。存储器分块结构如图 5-19(b)所示。

（4）典型的 NAND 闪存 Samsung K9XXG08UxM

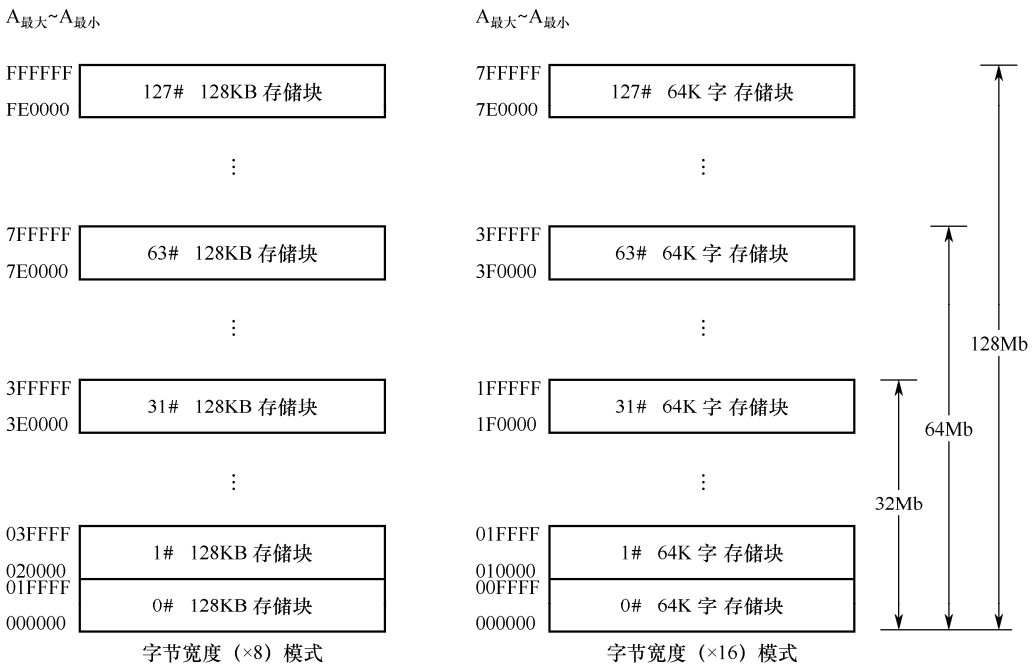
K9K8G08U0M、K9WAG08U1M、K9NBG08U5M 是 Samsung 公司所产的 1/2/8G×8b



(a) 结构图

图 5-19　32、64、128 MB 典型 NOR 闪存芯片 28F320/640/128J3

電子工業出版社

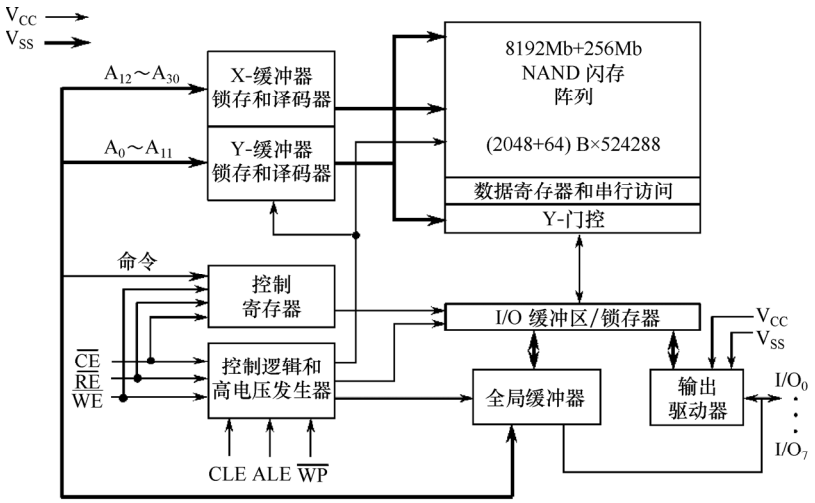


(b) 存储块图

图 5-19　32、64、128MB 典型 NOR 闪存芯片 28F320/640/128J3（续）

NAND Flash ROM。以 K9K8G08U0M 为例，其内部逻辑框图和主要引脚说明见图 5-20，具有以下特性参数：

- 1 GB+32 MB 存储体，有效存储空间 8032～8192 Mb，(2K+64)×8 b 数据寄存器。
- 支持自动编程和擦除模式，页编程(2K+64)B，块擦除(128K+4K)B。
- (2K+64)字节页，最大 20 μs 随机页面读/写周期，最少 25 ns 串行访问时间。
- 200 μs 快速页面编程周期，块擦除周期 1.5 ms。
- 具备硬件写保护功能，擦/写寿命 10 万次，信息保存寿命 10 年。



(a) 结构图

引脚名称	说　明
I/O_0～I/O_7	数据输入/输出
A_0～A_{30}	地址输入
CLE	命令锁存允许
ALE	地址锁存允许
$\overline{CE}$	片选
$\overline{RE}$	读允许
$\overline{WE}$	写允许
$\overline{WP}$	写保护
V_{CC}	电源
V_{SS}	地

(b) 引脚说明

图 5-20　典型 NAND 芯片 Samsung K9K8G08U0M 结构及引脚说明

電子工業出版社

5.4 存储器连接与扩充应用

在微机设计和应用中往往需要用一组存储器芯片构成一个存储系统，或根据需要扩充存储器容量。存储器提供指令代码和数据的信息存储功能，CPU 在取指令和执行操作数为存储器单元的指令的过程中访问相应的存储单元，需要在硬件电路上通过地址、数据及控制总线把存储器芯片与 CPU 连接起来，并使之协调工作。CPU 与存储器交换信息时一般总是先输出地址，再送出读/写控制信号，并通过数据总线进行信息交换。如前所述，每个存储器都有特定的读、写及擦除时序，读/写系统如 CPU 必须在时序上支持相应的操作过程。

半导体存储器中信息数据的正确读/写或擦除，每种芯片都有严格的性能规范。由于 CPU 和存储器往往根据各自的技术标准，由不同厂商生产，要保证正确的访问，除了要求连接电平标准相同，还必须考虑信号连接、时序配合等问题。

CPU 的负载驱动能力也是需要考虑的因素，因为在微机系统中，CPU 通过总线与存储器芯片、I/O 接口芯片连接，而 CPU 的总线驱动能力是有限的。但存储器芯片多为 MOS 电路，直流负载很小，主要负载为电容负载。在小型系统中，CPU 可直接与存储器芯片连接，但与大容量的存储器连接时就应考虑总线的驱动问题。

当 CPU 数据总线在 16 位以上，组合使用 8 位数据总线等存储器时，需要考虑读/写效率和逻辑允许等问题。有些 CPU 要求低位地址必须对齐（如 Intel 8096），有些在低位地址不对齐时需要多个读/写周期完成 16 位或 32 位操作，如 80x86 CPU。CPU 按字访问时，存储单元地址可以是偶数（A_0=0）或奇数（A_0=1），对于 16 位字长的 CPU 如 8086/80286，对偶数地址进行字访问时，地址是对齐时（A_0=0），访问效率高；对于 32 位数据总线的 CPU 如 80386/Pentium，则当地址对齐（A_1A_0=00）时，能够高效实现双字（32 位）数据访问。

5.4.1 存储器芯片选择

1. 存储器类型选择

如前所述，半导体存储器分成 RAM 和 ROM 两类。RAM 的特点是其存储的信息可以在程序中用读/写指令随机访问，但断电后信息丢失。所以 RAM 一般用于存储用户的调试程序（或程序存储器中的用户区）、程序运算的中间结果及断电后不需保护（存）的信息。

静态 RAM 在与 CPU 接口时，一般不需要外围电路，连接比较简单，故在智能仪器仪表、小型系统中，一般采用静态 RAM。

动态 RAM 集成度高，但需要专门的刷新电路，因此与 CPU 的接口设计较为复杂，但是由于成本相对较低，在需要较大存储器容量的计算机产品中得到了广泛应用。

ROM 中的内容断电后不丢失，但不能随机写入，故一般用于存放系统程序（监控程序）和不需在线修改的参数等。其中，掩膜 ROM 和 PROM 用于大批量生产的微电子产品或计算机产品中。在产品研制和小批量生产时，宜选用 EPROM 或 Flash ROM 等芯片，以便多次修改程序或用户自行编程。

E^2PROM 多用于保存这样一些数据或参数，它们在系统工作过程中被写入而又需要掉电保护。近年来串行接口的 E^2PROM 因其接口简单、工作可靠被大量用于系统设计，而 Flash ROM 被大量用于程序存储，尤其是支持软件的在线升级。

利用后备电源（一般是可充电电池）配合掉电保护电路，也可以使静态 RAM 在正常电

電子工業出版社

源掉电时数据不丢失，这时需要选择有双（高、低电平）片选的低功耗 RAM 芯片。

存储器系统设计先确定整机存储容量，再确定选用存储芯片的类型和数量，之后划分 RAM、ROM 区，画出地址分配图。存储器空间的划分和地址编码靠地址线来实现。多片存储芯片构成的存储器的地址编码原则是：低位地址总线作为片内寻址，高位地址总线将用来产生存储芯片的片选信号。需要设计专门的片选译码电路，详见 5.5 节。

2．连接接口电平

通常，CPU 总线与存储器连接引脚的电平标准应相同，同为 TTL、CMOS、LTTL、SSTL2、SSTL18 等标准电平，必要时需增加电平转换电路。

3．存储器芯片与 CPU 的时序配合

存储器的存取时间是反映其工作速度的重要指标，选用存储芯片时，需要分别从 CPU 和存储器芯片两个方面进行分析，保证存储器芯片读/写存取时间等工作时序和参数与 CPU 的时序和参数相匹配。

在 CPU 和存储器的数据手册中已设计规定了各自读/写操作时的地址、数据和控制总线信号的时序，因此连接时，必须注意它们的配合，如当 CPU 发出读数据信号时，存储器能输出数据并稳定在数据总线上，保证读操作能顺利进行。

典型的 8088 CPU 存储器读周期时序重画在图 5-21 中。对照表 5-6 并由图 5-21 可以看出，CPU 工作在 4.77 MHz 时钟频率时，从 T_1 状态开始到地址信号有效其最长时间 $TCLAV_{max}$=110 ns。由 5.2 节中的图 5-8 得出 HM6264BL 读取时间 t_{AAmax}=70 ns。于是从 T_1 状态开始到 HM6264BL 中指定单元读出信息到数据总线上的时间为 $TCLAV_{max}+t_{AAmax}$=110 ns+70 ns=180 ns

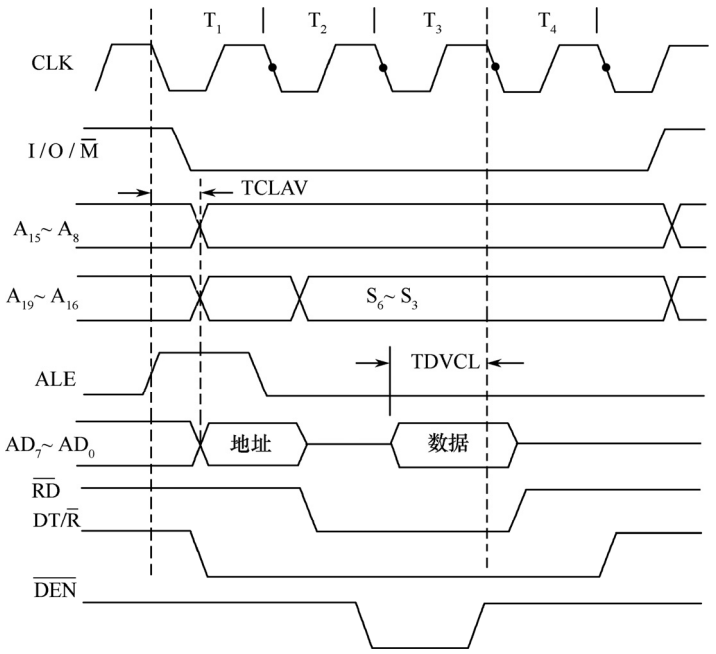


图 5-21　8088 CPU 存储器读周期时序

表 5-6　8088 CPU 存储器读周期时序参数

符号	TDVCL	TCLAV
参数	数据建立时间	地址有效延迟
最小/ns	30	10
最大/ns	—	110

8088 CPU 在 T_3 状态下跳沿采样数据总线得到数据。为确保采样数据是可靠的，要求

RAM 输出到总线上的数据比 T_3 后沿提前 TDVCL 时间达到稳定。从 T_1 开始到数据必须稳定的时间为 $3T$−TDVCL=3×200 ns−30 ns=570 ns。

而对于 HM6264BL 来说，实际上从 T_1 开始，只需经过 180 ns，数据已经读出至数据总线上，比 8088 CPU 所要求的 570 ns 大大提高。所以，8088 CPU 与 HM6264BL 存储器连接时，在存储器读周期，它们的时序能够很好地配合。

存储器写周期时序配合与读周期类似，不再重复叙述。如果存储器芯片读或写周期的工作速度不能满足 CPU 的要求，可在 CPU 的相应周期内插入一个或数个 T_W 周期，人为地延长 CPU 读/写时间，使两者相配。为简化外围电路及充分发挥 CPU 的工作速度，应尽可能选择与 CPU 时序相配的芯片。

5.4.2 存储器容量扩充

当一片存储器芯片的容量不能满足系统要求时，需多片组合以扩充字长（位扩充）或字数（字扩充）。下面以 SRAM 为例说明容量扩充的方法，ROM 的处理方法与此类似。

1．存储器位扩充

用 8K×8 b 的 SRAM 芯片 HM6264 扩充形成 8K×16 b 的芯片组，所需芯片为 16 b/8 b=2（片）。这两个芯片（0#和 1#）的地址线 A_0～A_{12} 分别连在一起，各芯片相应的片选信号及读/写控制信号也都分别连在一起，两个芯片只有数据线各自独立，一片连低 8 位（D_0～D_7），另一片连高 8 位（D_8～D_{15}），如图 5-22 所示。也就是说，每个 16 位数据的高、低字节分别存储于两个芯片单元中，一次读/写操作同时访问两个芯片的相同地址单元。

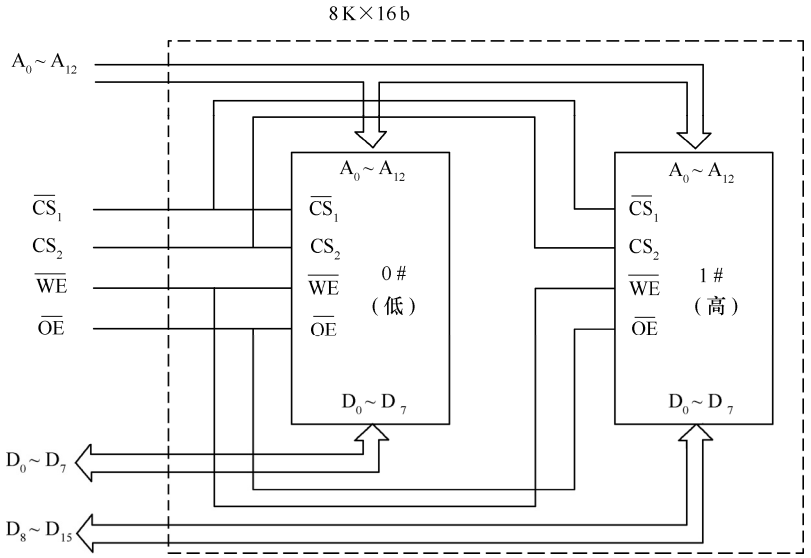


图 5-22　2 片 8K×8 b 芯片组成 8K×16 b 芯片组

2．存储单元数扩充

当存储器位数满足要求而需要扩充单元数时，也需要用若干芯片（组）组成新的芯片组，如用 8K×8 b 芯片 6264 构成 32K×8 b 的存储区时，所需要的 8K×8 b 芯片数为 32K/8K = 4（片）。

電子工業出版社

这 4 个 8 KB 芯片（0#～3#）连成 32K×8 b 芯片组，如图 5-23 所示，即各 6264 芯片的地址线 A_0～A_{12}、数据线 D_7～D_0 及读/写信号都是同名信号连在一起。字数的扩充使 32K×8 b 芯片组较 8K×8 b 芯片增加了两位地址信号 A_{13}、A_{14}，它们译码后产生 4 个片选信号，分别选中 4 个 8K×8 b 芯片中的一个，这 32K 的地址范围分配如表 5-7 所示。这里地址线 A_0～A_{12} 实现片内寻址，A_{13}～A_{14} 实现片间寻址。

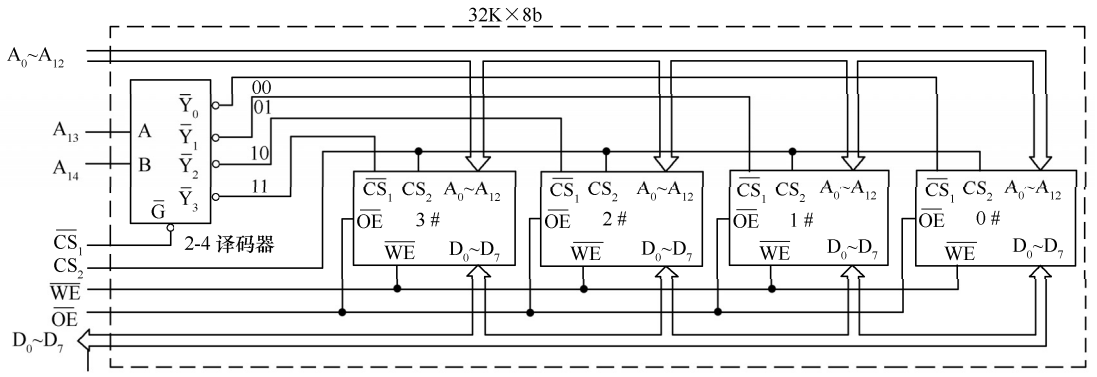


图 5-23　4 片 8K×8b 芯片扩充成 32K×8b 芯片组

表 5-7　4 片 8 KB 构成 32 KB 地址分配表

8K×8 b 芯片号	A_{14} A_{13}	A_{12}～A_0	地址范围（空间）
0#	0　0	00…0～11…1	0000H～1FFFH
1#	0　1	00…0～11…1	2000H～3FFFH
2#	1　0	00…0～11…1	4000H～5FFFH
3#	1　1	00…0～11…1	6000H～7FFFH

存储器的单元数和位数都需要扩充时，如用 8K×8 b 芯片构成 32K×16 b 存储区，需要 4×2 个芯片。可以先扩充位数，每 2 个芯片一组，构成 4 个 8K×16 b 芯片组；再扩充单元数，将这 4 个芯片组组合成 32K×16 b 存储块。

5.4.3　RAM 存储模块

高集成度 RAM 模块就是习惯上所说的内存条（Memory Module），其产生和发展与 CPU 性能的提高密切相关，利用 DRAM 基本存储电路存储信息，把动态刷新电路集成在片内，克服了 DRAM 需要加外部刷新电路的缺点，从而兼有静态和动态 RAM 的优点。

高集成度 RAM 将多片高容量 DRAM 芯片装配在条状印制线路板上，线路板配有标准单边或双边沿连接插脚，可直接插入微机主板的存储器插座，加上相应的控制电路，构成具有规定字长宽度及奇偶校验特性的存储器模块。

PC 系统常用的模块中，按其不同标准和数据字长，可分为 30 线（8+1 位）SIMM（Single In-line Memory Module）、72 线（32+4 位）SIMM 和 168 线、184 线、240 线（64+8 位）DIMM（Dual In-line Memory Module）等，并满足不同的总线速度。其中，30 线 SIMM 多用于 80386 以下系统，72 线 SIMM 多用于 80486 系统，168 线 DIMM 主要用于 Pentium 以上机型（PC66、PC100、PC133 等），而 PC 标准内存条 DDR SDRAM 184 线 DIMM、DDR2 和 DDR3 为 240 线（如图 5-24 所示），DDR4 为 288 线，主要应用参数包括前端总线频率、传输速率、存储容量等。

電子工業出版社

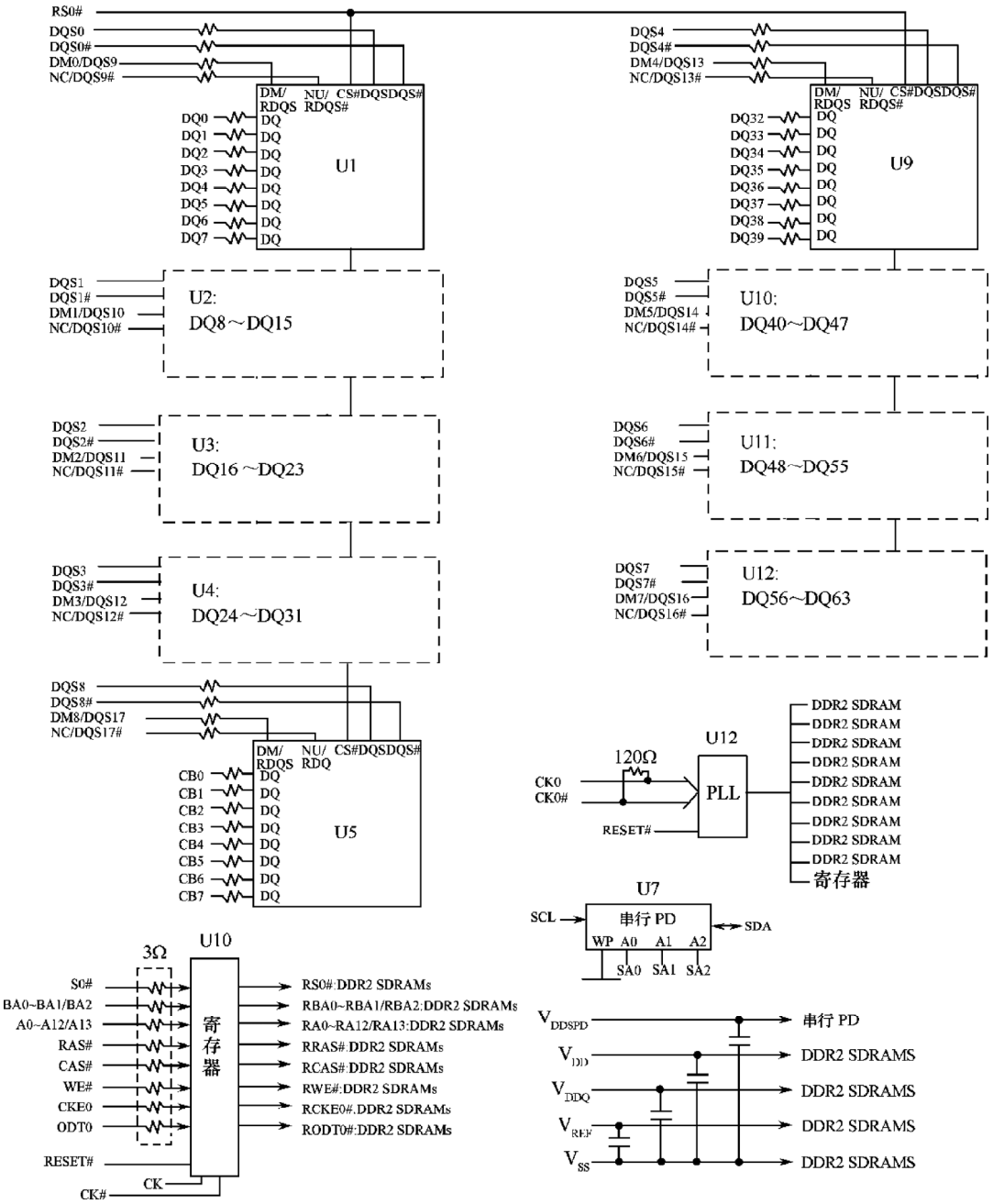


图 5-24　240 线 DIMM DDR2 模块 MT9HTF12872 原理

事实上，内存条是根据 CPU 的特性组合扩充得到的具有标准接口和读/写特性的存储模块。例如，30 线 LGS 产品 GMM79100 将二片 1M×4 b 和一片 1M×1 b（奇偶校验位）构成 1M×9 b 的 SIMM 模块，刷新频率为 1 kHz，5 V 供电，FPM 模式，存取速度为 60/70 ns；72 线 GMM7324210，以 8 片 4M×4 b 为基片，构成 4M×32 b DRAM SIMM，EDO 模式，5 V，刷新频率为 4 kHz，存取速度为 50/60 ns。

Pentium Ⅱ和 Pentium Ⅲ CPU 系统主要采用 168 线 SDRAM 模块，典型芯片如 Samsung 公司的 KMM375S1620BT，是一种容量 16M×72 b 的 SDRAM DIMM（128 MB 带奇偶校验），

上面集成了 18 片 16M×4 b 的 SDRAM 芯片，2K SPD（Serial Presentation Detect）串行 E^2PROM，以及锁相环 PLL 等辅助电路，工作电压为 3.3±0.3 V，LVTTL 输入/输出兼容，支持突发模式，自动及自刷新频率 44 kHz/64 ms，符合 PC100 标准，最高频率可达 125 MHz；支持 SPD 的母板可以从 E^2PROM 中读取 SDRAM 的工作参数，以避免由于参数设置不当而引起系统的不稳定。

Pentium 4 CPU 系统主要采用 DDR SDRAM，由一组 DDR SDRAM 存储器芯片组合集成，模块数据字长 72（64+8 位 ECC 校验纠错位），如 Micron 公司的 184 线 DDR 模块 MT18VDDT3272D 为 256 MB（32M×72 b）、MT18VDDT6472D 为 512 MB（64M×72 b）、MT18VDDT12872D 为 1 GB（128M×72 b）等，传输速率 PC1600/PC2400。240 线 DDR2 模块 MT9HTF3272 为 256 MB、MT9HTF6472 为 512 MB、MT9HTF12872 为 1 GB（128M×72 b），前端总线频率 400 MT（传输次）/s 和 533 MT/s，传输速率 PC2-3200 或 PC2-4200，其原理框图见图 5-24。

目前，微机系统多采用 DDR3 或 DDR4 存储器模块，其性能参数见 5.2 节的表 5-4。

DDR4 作为高速 SDRAM 内部配置成 16 个存储体、4 个存储体组带 4 个存储体其每组 x4/x8 及 8 存储体、2 个存储体组带 4 个存储体其每组 x16 DRAM。

5.5 CPU 与存储器的典型连接

5.5.1 8086/8088 CPU 的典型存储器连接

8086/8088 CPU 有 20 根地址线，其中低 16/8 位 AD_7（AD_{15}）～AD_0 与数据线复用，其余高位与状态位复用。例如 8088 CPU 与存储器相连时，须使用外部地址锁存器，由地址锁存信号 ALE 把 A_{19}～A_{16} 及 AD_7～AD_0 在地址锁存器上锁存，生成系统数据总线（如 PC 总线上的 A_{19}～A_0 就是锁存后的信号），然后（或驱动后）再连至存储器。

多数存储器与 8 位 CPU 兼容，即数据总线宽度为 8 位。下面以 8088 系统总线与静态 RAM 的连接为例，说明地址线、数据线和控制线的连接方法，特别注意地址分配和片选问题。例如，将 4 片 6264 芯片与系统总线相连构成 32K×8 b 存储区，数据线及控制信号的连接比较简单，如图 5-23 所示。地址线的连接较为灵活，一般地，存储器芯片的地址线 A_0～A_{12} 与系统对应的地址线 A_0～A_{12} 相连（片内寻址），系统高位地址信号 A_{13}～A_{19} 译码后产生各片 6264 的片选信号（片间寻址）。译码方式一般有 3 种。

1．全译码法

片内寻址未用的全部高位地址线都参加译码，译码输出作为片选信号，区分 4 片 6264（0#～3#），如图 5-25 所示，各片 6264 的地址范围如表 5-8 所示。整个 32K×8 b 存储器的地址范围为 00000H～07FFFH，仅占用了 8088 CPU 1 MB 存储容量的 32K 地址范围。

全译码的优点是每个芯片的地址范围唯一确定，且各片之间是连续的。缺点是译码电路比较复杂，一般可以用逻辑器件如 3-8 译码器或可编程逻辑器件等实现，如图 5-26 所示。

2．部分译码法

所谓部分译码，就是用片内寻址外的一部分高位地址参与译码，产生片选信号。图 5-27 是用部分译码法产生片选信号的原理图例。由于 4 片 6264 需要 4 个片选信号，因此用两位

電子工業出版社

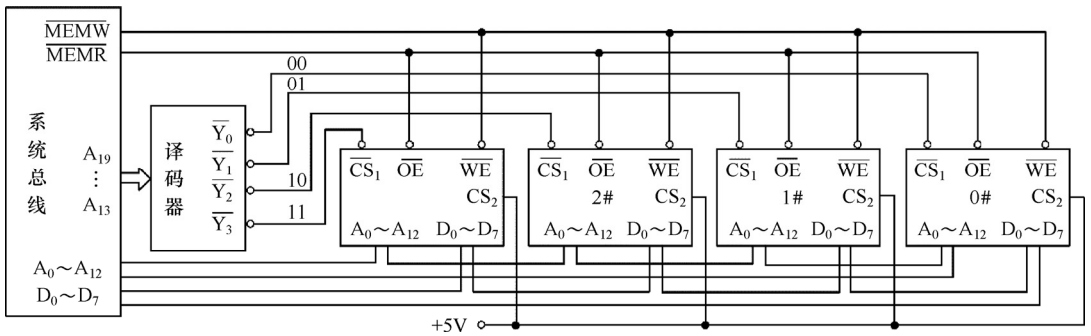


图 5-25　全译码实现存储器扩展

表 5-8　全译码方式下 4 片 6264 存储器地址

芯片	A_{19}～A_{15}	A_{14}　A_{13}	$A_{12}A_1\cdots A_0$	地址范围（空间）
0#	00…0	0　0	00…0～11…1	00000H～01FFFH
1#	00…0	0　1	00…0～11…1	02000H～03FFFH
2#	00…0	1　0	00…0～11…1	04000H～05FFFH
3#	00…0	1　1	00…0～11…1	06000H～07FFFH

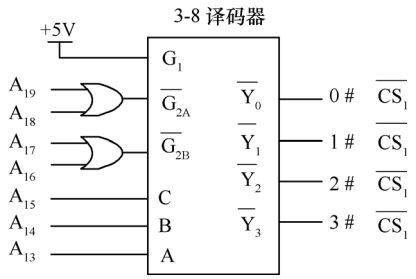


图 5-26　全译码电路的一种实现

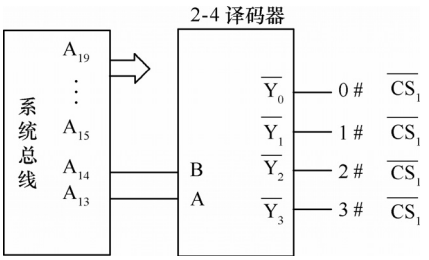


图 5-27　部分译码法实现存储器扩展

地址信号 A_{14}、A_{13} 来译码，而 A_{19}～A_{15} 不参与译码。其他信号（数据线、读/写信号）的连接同图 5-25。在图 5-27 中，由于译码器未用到高位地址 A_{19}～A_{15}，只要 A_{14}=A_{13}=0，无论 A_{19}～A_{15} 取何值，均选中第一片（0#）（其他片同理）。也就是说，32K RAM 中的任一个单元，都对应 2^{20-15}=2^5（32）个地址，这种一个单元有多个地址的现象称为地址重叠。从地址分布来看，这 32K 存储器实际占用了 8088 CPU 全部的 1 MB 存储容量，每片 6264 有 1M/4=256K 的地址重叠区。令未用到的高位地址信号全为 0，这样确定出的存储器地址称为基本地址，本例 32K×8 b 存储器的基本地址为 00000H～07FFFH。

可见部分译码较全译码简单，但存在地址重叠区。

3．线选法

线选法就是高位地址线不经过译码，直接分别接各存储器芯片的片选端来区别各芯片的地址。注意，必须保证这些片选线每次寻址时只能有一位有效（如本例中为低电平），不允许多于一位同时有效，才能保证硬件正常工作。

图 5-28 是用线选法产生 4 片 6264（0#～3#）片选信号的原理图例。A_{16}～A_{13} 用作片选，而 A_{19}～A_{17} 未用。其他信号（数据线、读/写信号）的连接同图 5-25。这时，存储器的 32K 基本地址范围如表 5-9 所示。线选法不仅会造成地址重叠，各芯片地址一般也是不连续的。

实际应用中，存储器芯片的片选信号可根据需要选择上述某种方法或几种方法并用。

电子工业出版社

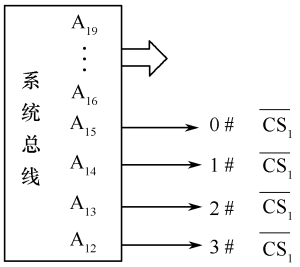


图 5-28　线选法实现存储器扩展

表 5-9　线选法 4 片 6264 存储器地址

芯片	A_{19}～A_{17}	A_{16}～A_{13}	A_{12}～A_0	地址范围
0#	000	0111	00…0～11…1	0E000H～0FFFFH
1#	000	1011	00…0～11…1	16000H～17FFFH
2#	000	1101	00…0～11…1	1A000H～1BFFFH
3#	000	1110	00…0～11…1	1C000H～DFFFH

ROM 芯片与 CPU 连接时，因 ROM 是只读的，故不需“写”信号。除此之外，ROM 与 CPU 连接时所要解决的问题（如容量扩充、地址译码等）及处理方法与 RAM 基本相同。

5.5.2　80386/Pentium CPU 的典型存储器连接

16 位、32 位及以上 CPU，为了向下兼容 8088/8086 等 CPU 指令，既要能读/写字节，还要具备 16 位及以上数据的读/写功能。在存储系统设计中，既要保证读/写功能，又要保证读/写时序的效率，这就要考虑存储单元的地址对齐问题。所谓地址对齐（Adjustment）指 16 位数据的存放地址为 2 的倍数（偶数地址，A_0=0），32 位数据的存放地址为 4 的倍数（A_1=A_0=0），64 位数据的存放地址为 8 的倍数（A_2=A_1=A_0=0），依次类推，以保证数据读/写效率，这种思想体现在 80386/Pentium 系统存储器的具体设计应用中。

8086 CPU 有专门的信号 BHE/S7（第 34 引脚），提供高 8 位数据总线的允许和状态信息复用引脚。BHE 可以看成一条附加的地址总线，用来访问存储器的高字节，A_0 用来访问存储器的低字节。所以，BHE 通常作为接在高 8 位数据总线上设备的片选信号，A_0 作为接在低 8 位数据总线上设备的片选信号。BHE 和 A_0 的状态和数据总线传送的数据关系见第 2 章。

80386 CPU 可通过 32 位数据总线实现 CPU 和存储器之间的数据传送。传输过程利用高 30 位地址线 A_2～A_{31} 加上 4 字节使能信号 $\overline{BE3}$～$\overline{BE0}$（即 BE3#～BE0#）来实现，形成字节、字或双字数据传输。$\overline{BE0}$～$\overline{BE3}$ 信号由 CPU 内部根据地址线 A_0、A_1 产生。典型连接如图 5-29 所示。存储系统设计保证了 4*n*+0 地址单元中的数据字节对应数据总线上双字数据的 D_7～D_0，4*n*+1 地址单元字节对应 D_{15}～D_8，4*n*+2 地址单元字节对应 D_{23}～D_{16}，4*n*+3 地址单元对应 D_{31}～D_{24}。

Pentium CPU 内部寄存器字长 32 位，通过片内高速缓存形成 64 位数据总线，因而主存储器通常按 64 位来构造，这就意味着，总是以 8 的整数倍地址实现物理寻址，以实现 CPU 与内存间的字节、字、双字或四字数据的传送。

Pentium CPU 有专门的存储体允许引脚信号 $\overline{BE_7}$～$\overline{BE_0}$（即 BE7#～BE0#）用于选择单字节、字、双字或四字数据读/写。这些信号在 CPU 内部的逻辑根据地址 A_0、A_1、A_2 产生，其连接图与图 5-29 类似，即：

- 8*n*+0 地址单元中的数据字节对应数据总线上 4 字数据的 D_7～D_0。
- 8*n*+1 地址单元中的数据字节对应数据总线上 4 字数据的 D_{15}～D_8。
- …
- 8*n*+7 地址单元中的数据字节对应数据总线上 4 字数据的 D_{63}～D_{56}。

電子工業出版社

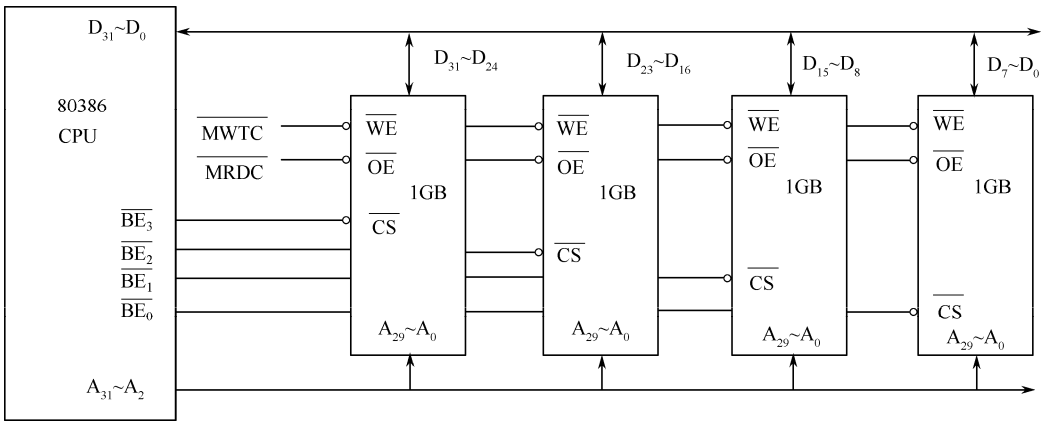


图 5-29　80386 CPU 存储体选择电路

这时传送 4 字只需要一个总线周期，但实际指令中的 4 字访问也可以非 8 的倍数作为起始地址，不过此时不能一次完成读/写这类地址的数据。Pentium 系列 CPU 将其拆分为两次总线访问，第 1 次读/写 4 字数据的低位字节部分，第 2 次读/写其高位字节部分，这种操作称为 4 字边界（Quad Word Boundary）。在这种情况下，两次传送的数据总数为 16 字节，但有效数据只有 8 字节。

例如，当用一个非 8 整数倍地址 203A23H 寻址 4 字时，总线接口按下述过程（效果）执行 2 次访问操作。① 读存储单元 203A23H～203A27H 中的 5 字节，CPU 产生的地址是 203A20H，所以必须将字节使能信号设置为 $\overline{BE_7}$ ～ $\overline{BE_3}$ 为 0、$\overline{BE_2}$ ～ $\overline{BE_0}$ 为 1，以便有效读取 4 字数据低位部分的 5 个有效字节。② CPU 给出地址 203A28H 并同时设 $\overline{BE_7}$ ～ $\overline{BE_3}$ 为 1，$\overline{BE_2}$ ～ $\overline{BE_0}$ 为 0，以便有效读取 4 字数据高位部分的 3 个有效字节，于是两次访问所获取的数据即为所需的 4 字。写操作的过程类似。

总而言之，在每次访问过程中，存储器控制器每次都输出全部 64 位数据到数据总线上，但 CPU 只接收 $\overline{BE_x}$（x=0～7）为低电平时所对应的那些字节。

为了提高系统的操作效率，在程序设计应用中得出以下存储变量地址边界对齐的一般结论：① 字节访问使用任意地址；② 单字访问使用偶数地址；③ 双字访问使用 4 倍地址；④ 4 字访问使用 8 倍地址，以此类推，否则本来一次访问就可完成的操作将被迫拆分为两次操作。

由此可知，为了提高数据操作的效率，在存储器地址分配时尽量使变量地址边界对齐，必要时需进行调整。

5.6　微机系统的内存结构

随着技术的发展，存储器的综合性能指标在不断提高，同时系统的总体性能和需求在不断提高。除了一般意义下使用内部存储器技术，现代微机系统越来越多地采用分级存储技术，通过引入高速缓存和虚拟存储器提高系统的速度，扩大存储空间，实现高速处理和多任务、多用户系统应用。

目前，微机系统一般采用三级存储器体系结构，即高速缓冲存储器、内部存储器和外部（辅助）存储器。CPU 可以直接访问高速缓存和内存，但不能直接访问外部存储器。

電子工業出版社

外部（辅助）存储器除了用作海量信息存储，还为虚拟存储器的实现提供了手段。尤其在中高档微机系统中，为了支持多用户、多任务运行环境，大容量虚拟存储器需利用外部（辅助）存储器实现。Intel 架构 32 位 CPU（IA-32）虚地址保护模式为虚拟存储器的资源管理和保护提供了专门的分段、分页和保护软/硬件技术。

5.6.1 分级存储结构

在半导体存储器中，只有双极型静态 RAM 的存取速度与 CPU 速度处于同一数量级，但其价格较贵、功耗大、集成度低，体积相对 DRAM 更大，因此不可能所有的存储器都采用静态 RAM，这就需要设计一种分级管理机制，在主存和 CPU 之间加一个容量相对小的双极型静态 RAM，作为高速缓冲存储器（简称 Cache），并实行专门的控制。

另一方面，现代微机系统对内存空间的要求越来越大，内/外存储器存在着的巨大差异，而多用户、多任务运行环境在客观上允许程序和数据对内存资源分时复用，在这种情况下，通过特殊的存储器管理机制，可以把处理器直接“透明”访问的存储空间扩大到整个虚拟存储器。

现代微机系统通常采用如图 5-30 所示的三层分级存储结构：① 高速缓存 Cache，由静态 RAM 构成；② 主内存，由动态 RAM 构成；③ 虚拟存储器，由硬盘等构成。对于使用最频繁的容量不大的程序和数据用高速缓存 Cache 存放，对经常使用的数据或程序存放于主内存中，而对不常用或暂时不用的大部分程序和数据就存放在磁盘（虚拟存储器）中。对内存的这种多层次的管理方式，可以有效地解决速度和成本之间的矛盾。

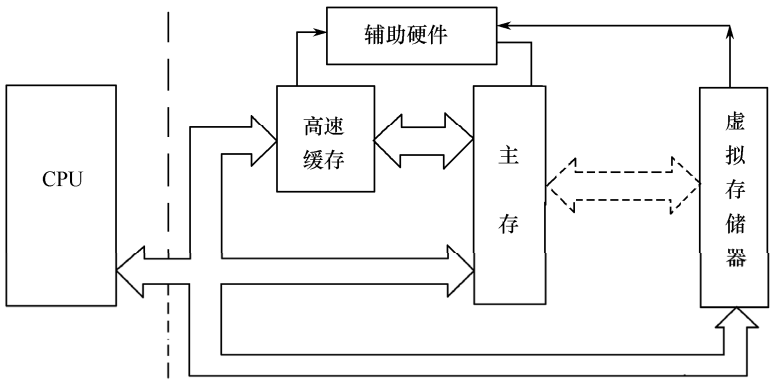


图 5-30 主存-Cache-虚拟存储器分层结构

5.6.2 高速缓存 Cache

高速缓存 Cache 的引入是为了减小乃至消除 CPU 与内存之间的存取速度差异对系统性能带来的影响。Cache 中一般保存有一份内存中部分内容的副本，该部分内容是最近曾被 CPU 使用过的数据和程序代码。Cache 的有效性是利用了程序对存储器的访问在时间和空间上所具有的局部区域性，即对大多数程序来说，在某个时间片内会集中重复地访问某一个特定的区域。同时，在正常情况下，访问 Cache 的速度要比访问内存的速度快 3～8 倍，所以 Cache 的使用可缩短 CPU 访问存储器的时间，从而提升系统的整体运行效率。

Cache 存储器位于主存和 CPU 之间，容量较小，由静态 RAM 构成。微机采用 Cache

電子工業出版社

控制器来协调 CPU 和 Cache、主存之间的数据传输，CPU 不仅与 Cache 相连，与主存也保持通路，如图 5-31 所示。

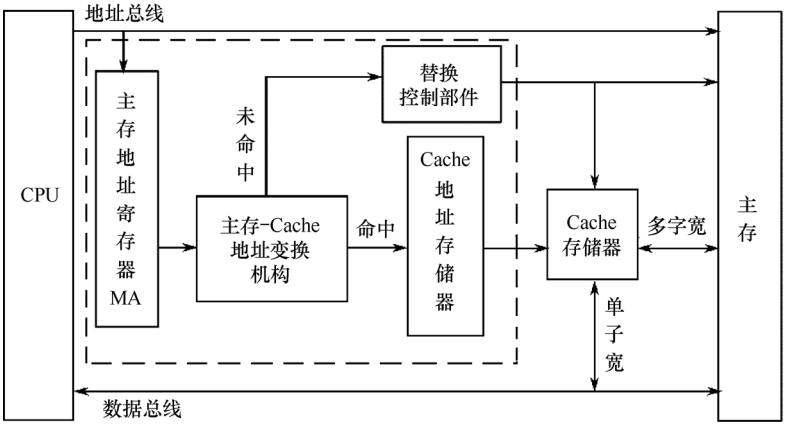


图 5-31　Cache 存储系统基本结构

Cache 控制器（虚框内）将来自 CPU 的数据读/写请求转给 Cache 存储器，如果数据已在 Cache 中，则一次命中，CPU 对 Cache 进行操作，这时 CPU 从 Cache 中读（写）数据。由于 Cache 速度与 CPU 速度相匹配，因此不需要插入等待状态，故 CPU 处于零等待状态，也就是 CPU 与 Cache 达到了同步。若数据不在 Cache 中，即一次未命中，则 CPU 对主存操作，CPU 必须在其机器周期中插入等待周期。目前 Cache 存储器容量可达 512KB 以上，这些大容量的 Cache 存储器，使 CPU 访问 Cache 的命中率高达 90%～98%，大大提高了 CPU 访问数据的速度，提高了系统的性能。

现代微机系统进一步将高速缓存分为 CPU 片内 Cache 和 CPU 外部（板级）Cache 两级，从而提高了命中率，加快了处理速度，使 CPU 对 Cache 的操作命中率高达 98%。

对程序员来说，Cache 是透明的。所谓“透明”，是指程序员自己不能对 Cache 进行操作和控制。Cache 由一个目录（或称为标志）及一个数据存储器组成。每当 CPU 需读/写数据时，它首先访问标志存储器，以确定会不会命中 Cache 以及 Cache 内是否有所需的信息。如果 Cache 标志不符合，说明 Cache 内没有所需的数据字，这种情况就是 Cache 不命中。如果 Cache 标志符合，就可以对 Cache 数据存储体进行读/写操作。但未命中时，对内存访问可能比访问无 Cache 的内存要插入更多的等待周期。而程序中的调用和跳转指令会造成非区域性操作，也会导致命中率降低。因而，提高命中率是设计 Cache 的主要目标。

为了便于替换和管理，通常把 Cache 和内存等分成相同大小的块（或行），每一块由若干字（或字节）组成。每当对一个内存地址进行访问时，都必须通过“内存-Cache”地址映像变换机构判定该访问字所在的块是否已在 Cache 中。如果在 Cache 中（Cache 命中），则经地址映像变换机构将内存地址变换成 Cache 地址去访问，这时 Cache 与 CPU 之间进行单字宽信息的交换；如果不在 Cache 中，则产生 Cache 块失效，这时需要从访问内存的通路中把包含该字的信息一起通过多字宽通路调入 Cache，同时将被访问字直接从单字宽通路送往 CPU。Cache 中原来的内容就会被覆盖掉一部分。置换算法通常有“先进先出（FIFO）”和“最近很少使用（LRU）”两种，通常在置换控制器中由硬件逻辑实现。

从内存将某些内容调入高速缓存，通常是以“页”为单位进行的，如以 256 字节为一页。高速缓存中各页的位置与内存中相应页的映像关系，决定了对高速缓存的管理策略，

電子工業出版社

也影响 Cache 的命中率。主要的映像关系有以下 3 种。

① 全关联方式：Cache 和内存均分为若干字节数相同的页。内存中的任一页都可调入 Cache 的任一页中，所调入页的页号需全部存入地址索引机构。寻址时将寻址地址同标志地址（页号）进行比较。

② 直接映射方式：Cache 中全部单元被划分成固定大小的页，内存先被划分成段，再被划分成与 Cache 大小相同的页。Cache 中的各页只接收内存中相同页号的内容，地址索引机构中存放的标志地址是内存的段号，而不是页号。寻址时只比较段号，不需比较页号，减少了地址比较次数。

③ 分组关联方式：前两种方式的折中，Cache 和内存都分为若干对应的组，然后组内直接映射、组间全关联映射，允许不同段的相同页号的内容同时存放在 Cache 中。

5.6.3 虚拟存储器与段页结构

虚拟存储器采用硬件和软件的综合技术，将主存（内部存储器）和辅存（外部存储器）的地址空间统一编址，形成一个庞大的存储空间。在这个存储空间里，应用程序编程运行，完全不必考虑主存是否装得下或者放在辅存的程序装入到主存中的实际位置。

目前，多用户、多任务软件多使用虚拟内存，如 Windows、Linux 等操作系统。具有专门的存储器管理单元（MMU）的 CPU 基本上提供专门的管理机制，尤其是 Intel 80386 以后的 CPU 都提供虚地址保护模式，专门用于虚拟存储器程序的运行和数据处理，提供相应的分段与分页机制。

由第 2 章可知，80386/Pentium CPU 保护模式下 MMU 使用 48 位存储器指针，指针由两部分组成：选择符和偏移量。选择符 16 位长，偏移量 32 位长，如指针 1230:9ABC5678H。该指针称为虚拟地址，在程序中指定指令或数据的存储位置。

其中，选择符放在 CPU 的段选择符寄存器中，指针的这一部分实际上选择的是 CPU 虚拟地址空间中一个特定的段。偏移量放在 CPU 用户可访问的寄存器中，或用于直接寻址。如进行指令代码的访问（取指），偏移量可放在 EIP 寄存器中，指针提供要访问的存储器段中的偏移量，即要执行的指令码的开头 1 字节的相对偏移地址。由于偏移量长 32 位，段空间可达 4 GB。事实上，段的大小是可变的，可以是 1 B～4 GB。

在分段机制中，系统和应用程序可用的虚拟地址空间为 64 TB，空间分成了 32TB 的全局存储器地址空间和 32 TB 的局部存储器地址空间，但是 80386 等 32 位 CPU 在 32 位模式下，地址总线只支持 4 GB 的物理地址空间，因此同一时间只有虚存的一部分信息可以放在物理存储器中，需要动态交换并有效管理。

事实上，Intel 80386/Pentium CPU 支持分段和分页两种存储管理机制，可以将 48 位虚拟地址映射到硬件所需的 32 位物理地址。分段机制下每个存储块大小不一，可以在 4 GB 范围内自行定义，每个段最多可以分配 4 GB 的物理地址空间，最少 1 字节。物理空间多次交换后可能变得非常零碎，因此存储和管理的对象较为复杂。分页机制管理下的对象是固定大小的存储块，如 4 KB（80386 及以上）或 4 MB（Pentium 及以上）等，称为页（Page）。这时把整个线性地址空间和整个物理地址空间都看成由很多页组成，线性地址空间的任何一页都可以映射到物理地址空间的任何一页。分页机制提供适合虚拟存储管理的存储块大小，避免了分段机制管理的复杂性。

電子工業出版社

有关 80386/Pentium CPU 分段/分页机制的具体原理详见第 2 章。

习 题 5

5-1　存储器是如何分类的？内存和外存各有什么特点？

5-2　RAM 和 ROM 各有何特点？静态 RAM 和动态 RAM 各有何特点？

5-3　如何判断有无地址重叠？有地址重叠时会出现什么问题？程序上应如何配合？

5-4　若存储空间的首地址为 1000H，写出存储器容量分别为 1K×8、2K×8、4K×8、8K×8 位时所对应的末地址。

5-5　试设计一片容量为 32K×8 b 的 EPROM 芯片与 8088 CPU 连接方案。写出设计原理，画出连线图。假定 EPROM 芯片所占地址空间（设起始地址 20000H）。

5-6　某系统的存储器配备了两种芯片：容量分别为 2K×8 b 的 EPROM 和容量为 1K×8 b 的 RAM。采用 74LS138 译码器产生片选信号：$\overline{Y}_0$、$\overline{Y}_1$、$\overline{Y}_2$ 直接连到三片 EPROM（1#、2#、3#）；$\overline{Y}_4$、$\overline{Y}_5$ 则通过一组门电路产生 4 个片选信号接到 4 片 RAM（4#、5#、6#、7#）。如图 5-32 所示，试分析电路原理，确定每一片存储器的寻址范围，并写出分析过程。

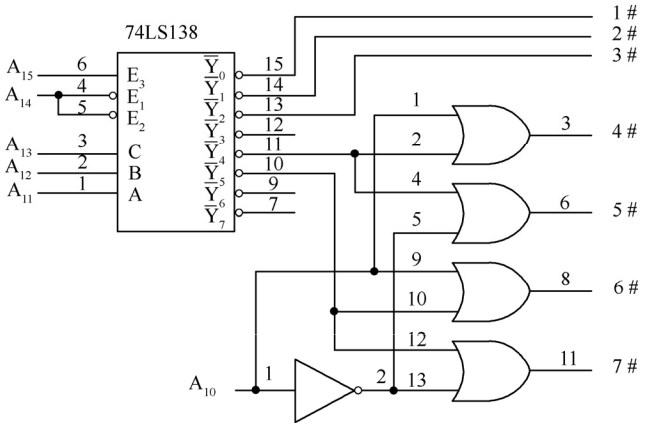


图 5-32　习题 5-6 图

5-7　试用 HM6116 芯片（SRAM，2K×8 b）组成 8K×8 b 的 RAM 系统，要求说明设计原理，画出它与 8088 CPU 的连线图（设起始地址为 80000H），说明设计过程。

5-8　试设计 62256 芯片（32K×8 b）与 8088 CPU 相连接方案，说明设计原理，画出连线图，设起始地址为 40000H。

5-9　了解并简述 IBM PC/XT 机的存储器子系统的构成。

5-10　了解并说明新型存储器中的存储体概念和特点。

5-10　查阅 DDR SDRAM 存储器模块（如 5.2.3 节介绍的型号）的数据手册（Datasheet），了解并说明它的基本原理和性能。

5-11　简述分段和分页机制原理。

電子工業出版社

第 6 章　输入/输出和中断

6.1　输入/输出及接口

输入和输出设备（称为 I/O 设备或外设）是计算机系统的重要组成部分，计算机通过它们与外界进行数据交换。如程序、原始数据及各种现场采集得到的信息，都必须通过输入装置输入到计算机；计算机也必须把计算的结果或各种控制信号送到各输出装置，以便显示、打印和实现各种控制动作。计算机常用的 I/O 设备有键盘、鼠标、显示器和打印机等。

6.1.1　I/O 信息的组成

在计算机控制系统中，外设与 CPU 之间需频繁交换的信息有以下三种。

（1）数据信息

CPU 与外设间交换的基本信息就是数据信息，分为数字量、模拟量和开关量三种形式。

① 数字量：按一定的编码标准（如以二进制格式或 ASCII 码标准），由若干位组合表示的数或字符。如 CPU 与键盘或打印机等 I/O 设备交换的信息就属于数字量。

② 模拟量：在计算机控制系统中，大量的现场信息经过传感器把非电量（如温度、压力、流量等）转换成电量，并经放大处理得到模拟量的电压或电流。这些模拟量必须先经过 A/D 转换器的转换才能输入计算机；计算机控制数据的输出也须经过 D/A 转换器转换成模拟量才能去控制执行机构。

③ 开关量：一些开关状态或控制量，如开关的断开与闭合等。通常，这些开关量要经过相应的电平转换才能与计算机连接。开关量只要一位二进制数即可表示，故对字长为 8 位（或 16 位）的计算机，一次可输入或输出 8 位（或 16 位）开关量。

数据的传送可采用并行传送和串行传送两种形式，本章只讨论并行传送。

（2）状态信息

状态信息指在 CPU 与外设之间交换数据时的联络信息。CPU 通过对外设状态信息的读取，可得知其工作状态。如了解输入设备的数据是否准备好，输出设备是否空闲等。因此，了解状态信息是 CPU 与 I/O 设备之间进行正确的数据交换的重要条件。

（3）控制信息

控制信息指 CPU 发给外设的命令信息，如设置 I/O 设备工作模式的信息等。CPU 通过发送控制信息控制外设的工作。

6.1.2　I/O 接口概述

接口是连接计算机和外设的部件，通常指接口电路，广义的接口还包括管理接口电路的驱动程序。

電子工業出版社

1．I/O 接口的功能

理论上，CPU 与 I/O 设备之间的连接及信息处理和 CPU 与存储器之间的连接及信息处理类似。实际上，I/O 设备类型繁多，工作速度高低不一，传输数据的字长多样，CPU 与其交换信息相应复杂。因此，I/O 设备通常并不直接与 CPU 交换信息，而是通过相应的输入/输出接口（称为 I/O 接口）来进行。I/O 接口是 CPU 和 I/O 设备之间进行信息交换的中间环节。

在实际应用中，I/O 接口电路种类繁多，功能各异，可分为专用接口和通用接口，或分成可编程接口和不可编程接口，也可分成并行接口和串行接口，其主要功能有：① 对输入/输出数据进行缓冲、隔离和锁存；② 完成信息格式的变换，如串-并转换等；③ 实现电气特性的匹配；④ 选择设备或地址译码等。

2．I/O 接口的构成

当 CPU 通过 I/O 接口与外设交换数据、状态和控制信息时，一般是通过接口电路中的寄存器来实现的，这些寄存器分别称为数据、状态和控制端口。每个 I/O 接口电路通常包含若干这样的寄存器端口，CPU 通过对这些端口的读/写实现与外设的信息交换。

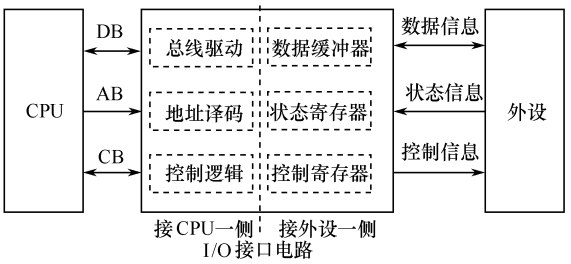


图 6-1　I/O 接口电路的典型结构

I/O 接口电路的典型结构如图 6-1 所示。其中，总线驱动实现对 CPU 数据总线速度和驱动能力的匹配；地址译码实现对外设中各寄存器端口的寻址操作；控制逻辑实现 CPU 对接口电路中各寄存器端口的读/写操作和时序控制。需要强调的是，状态信息和控制信息通常也是通过数据总线传送的，由于它们的性质不同于数据信息，在传送时被赋予不同的端口。因此，一个外设往往占用几个端口，这样一来，CPU 对外设的控制或 CPU 与外设间的信息交换，实际上转换成了 CPU 通过 I/O 指令进行端口的操作。只是对不同的端口，读/写的数据性质不同罢了。

3．I/O 接口的类型

早期的计算机外设种类非常有限，相应的接口标准也很简单。通常使用 PS/2 接口连接鼠标和键盘，串行接口（RS-232）连接外置 Modem，并行接口接入针式打印机或扫描仪等。

随着计算机应用技术的发展，外设的丰富性得到了提高，现在的计算机增加了许多新型接口，如 USB 接口、IEEE1394 接口、VGA 接口和 RJ-45 接口等。其中，USB 接口因其具有非常多的优点，使它能够取代几乎所有现有的串行接口。鼠标、键盘、移动硬盘、扫描仪或打印机等一切带 USB 接口的外设都可由 USB 接口连接到 PC 上；IEEE1394 接口适用于一些传输数据量较大的设备，如数码摄像机；VGA 接口是专用于连接显示器的接口；RJ-45 接口用来接入局域网或连接 ADSL 等上网设备，取代了早期的 Modem。

6.1.3　I/O 端口的编址

由于 CPU 与外设间的信息交换是通过接口中的 I/O 端口来完成的，因此 CPU 必须对 I/O 端口进行寻址。CPU 寻址外设端口一般有两种方式。

電子工業出版社

1．I/O 端口独立编址

I/O 端口独立编址也称为直接 I/O 映射的 I/O 编址，这时系统的存储器地址空间和 I/O 端口地址空间为两个不同的独立地址空间。CPU 通过不同的指令和专门的控制信号来区分是访问存储单元还是 I/O 端口。这种方法的优点是：由于使用了专门的 I/O 指令，容易分清指令是访问存储器还是访问外设，程序易读性较好；又因为 I/O 端口的地址空间独立且一般小于存储空间，所以其控制译码电路相对简单。

2．I/O 端口与存储器统一编址

I/O 端口与存储器统一编址方式也称为存储器映射的 I/O 编址，把存储器单元和外设端口统一编址，CPU 用同样的指令对存储器单元和 I/O 端口进行访问。该方法的优点是：无须专门的 I/O 指令，端口寻址手段丰富，相互之间依靠地址的区域不同加以区分。但由于外设端口占用一部分地址空间，使存储器能够使用的地址空间减小，在程序中不易分清哪些指令是访问存储器的，哪些指令是访问外设的，程序的易读性会受到影响。

80x86/Pentium CPU 采用 I/O 端口独立编址方式，端口地址 16 位。与存储器地址空间相似，I/O 地址空间在物理上也由 1、2、4、8 个 8 位 I/O 体构成，以支持 8、16、32、64 位 I/O 端口操作。因而，16 位地址线可寻址 64K 个 8 位端口、32K 个 16 位端口、16K 个 32 位端口或 8K 个 64 位端口。但在实际的微机系统中，往往只使用一部分地址线进行 I/O 端口地址译码。如在 PC/XT 中，实际参与端口寻址的只有低 10 位地址线 A_9～A_0。因此 PC/XT 的 I/O 端口空间的大小为 1 KB，以 A_9 来区别端口所在位置。当 A_9=0 时，寻址主板上的 512 个端口；A_9=1 时，寻址 I/O 扩展卡上的 512 个端口。PC/XT 主板上设备的 I/O 译码电路如图 6-2 所示，译码的条件是 AEN 为低电平（高电平时，表示由 DMA 控制器送出的地址有效），表示 CPU 掌控总线。PC/XT 和 PC/AT 机的 I/O 端口地址分配情况如表 6-1 所示。

80386 及以上 CPU 的 I/O 系统在 I/O 端口地址分配上完全与早期的 Intel CPU 兼容，均可寻址 64 KB 地址空间。80386 和 80486 CPU 是由地址线 A_{15}～A_2 寻地址，并用 $\overline{BE_3}$ ～ $\overline{BE_0}$ 选择传输 8 位、16 位或 32 位的 I/O 数据。而 Pentium 系列 CPU 的 I/O 系统，由地址线 A_{15}～A_3 和 $\overline{BE_7}$ ～ $\overline{BE_0}$ 一起选择 I/O 体。表 6-2 显示了 Pentium 系列微机系统中部分 I/O 端口的地址空间的分配。

用户在设计扩展接口时，需注意不要使用系统已占用的地址空间。表中未指明的端口地址是最可靠的用户可用地址。

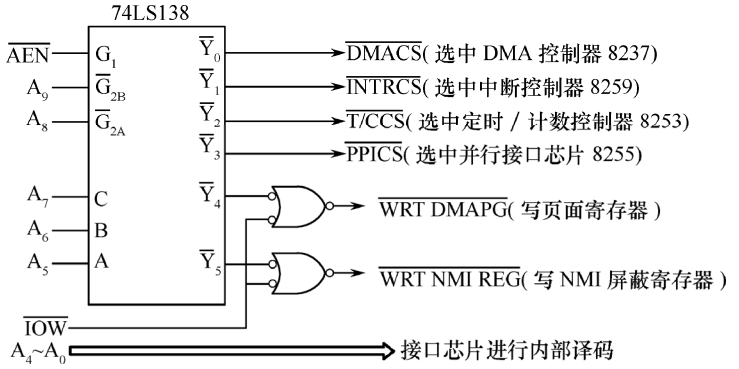


图 6-2　PC/XT 主板上设备的 I/O 译码电路

電子工業出版社

表 6-1　PC/XT、PC/AT 机的 I/O 端口地址分配

分类	PC/XT		PC/AT	
	地址范围	I/O 设备	地址范围	I/O 设备
系统板	000～01FH	DMA 控制器 8237A	000～01FH	DMA 控制器 1
	020～03FH	中断控制器 8259A	020～03FH	中断控制器 1
	040～05FH	定时器/计数器 8253	040～05FH	定时器/计数器
	060～07FH	并行接口电路 8255A	060～07FH	并行接口电路
	080～09FH	DMA 页面寄存器	080～09FH	DMA 页面寄存器
	0A0～0BFH	NMI 屏蔽寄存器	0A0～0BFH	中断控制器 2
	0C0～1FFH	保留	0C0～0DFH	DMA 控制器 2
			0F0H	清除协处理器忙信号
			0F1H	复位协处理器
			0F8～0FFH	数学协处理器
I/O 通道	—	—	100～16FH	保留
	—	—	170～1F8H	硬盘适配器 2、1
	200～20FH	游戏接口	200～207H	游戏接口
	210～217H	扩展箱	—	—
	220～2F7H	保留	278～27FH	并行打印机接口 LPT2
			2E8～2EFH	串行通信接口 COM4
	2F8～2FFH	串行通信接口 COM2	2F8～2FFH	串行通信接口 COM2
	300～31FH	试验卡	300～31FH	试验卡
	320～32FH	硬盘适配器	—	—
	—	—	360～36FH	保留
	—	—	370～377H	软盘适配器 2
	378～37FH	并行打印机接口 LPT	378～37FH	并行打印机接口 LPT1
	380～38FH	SDLC 通信接口	380～38FH	SDLC 通信接口
	3A0～3AFH	保留	3A0～3AFH	BSC 通信接口
	3B0～3BFH	单色显示/打印机适配器	3B0～3BFH	单色显示/打印机适配器
	3C0～3CFH	保留	3C0～3CFH	保留
	3D0～3DFH	彩色图形适配器 CGA	3D0～3DFH	彩色图形适配器 CGA
	3E0～3E7H	保留	—	—
	—	—	3E8～3EFH	串行通信接口 COM2
	3F0～3F7H	软盘适配器	3F0～3F7H	软盘适配器 1
	3F8～3FFH	串行通信接口 COM1	3F8～3FFH	串行通信接口 COM1

表 6-2　Pentium 系列微机中部分 I/O 端口地址空间的分配

地址范围	XX0H～XX3H	XX4H～XX7H	XX8H～XXBH	XXCH～XXFH
000～00FH	DMA 控制器			
020～02FH	中断控制器 1	系统使用		
040～04FH	系统时钟	系统使用		
050～05FH	系统使用			
060～06FH	键盘和鼠标口			
070～07FH	RTC/CMOS，NMI	系统使用		
080～08FH	DMA 页面寄存器 0～7			
0A0～0AFH	中断控制器 2	系统使用		
0C0～0DFH	DMA 控制器			
130～15FH	SCSI 主适配器			
160～16FH			IDE 控制器 4	
170～17FH	IDE 控制器 2			

電子工業出版社

续表

地址范围	XX0H～XX3H	XX4H～XX7H	XX8H～XXBH	XXCH～XXFH
1E0～1EFH			IDE 控制器 3	
1F0～1FFH	IDE 控制器 1			
200～20FH	游戏接口			系统使用
220～22FH	声卡			
230～23FH	SCSI 主适配器			
240～24FH	声卡			
250～2BFH	以太网卡（即插即用系统设备：274～277H，LPT2，LPT3：278～27FH）			
2E0～2EFH			COM4	
2F0～2FFH			COM2	
300～31FH	声卡或以太网卡			
320～32FH	硬盘适配器			
330～36FH	SCSI 主适配器，声卡或以太网卡			
370～37FH	软盘适配器		LPT1，LPT2	
3B0～3DFH	VGA/EGA/CGA 图形适配器			
3E0～3EFH	软盘适配器		COM3	
3F0～3FFH	游戏接口		COM1	

6.1.4 简单的 I/O 接口

不同的 I/O 设备所需采用的 I/O 接口电路的复杂程度相差甚远，但分解到最基本的功能，接口中应用最多的是三态缓冲器和数据锁存器。

1．三态缓冲器

在微机系统中，每个输入设备都需通过数据总线向 CPU 传送数据，若不经过三态门电路进行缓冲隔离而直接和数据总线相连，就会造成总线上数据的混乱。因此，每个接入设备必须经过缓冲隔离，当 CPU 选通时，才允许选定的输入设备将数据送到系统总线，其他的输入设备此时与数据总线隔离。

74LS244 是一种常用的三态缓冲器，其引脚及内部结构如图 6-3 所示，可以构成简单的输入接口，如图 6-4 所示。8 个开关 K_1～K_8 接 74LS244 的输入，其输出接到系统的数据总线上。当对此端口进行输入操作时，地址译码和 $\overline{IOR}$ 同时有效，于是 $\overline{1G}$ 、$\overline{2G}$ 有效，8 个开关的开合状态所表示的二进制数被输入到计算机中。

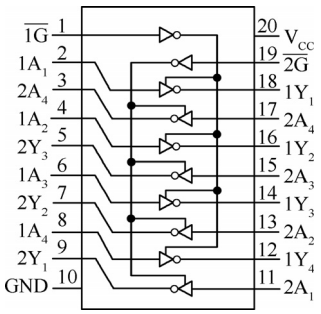


图 6-3　74LS244 引脚及内部结构

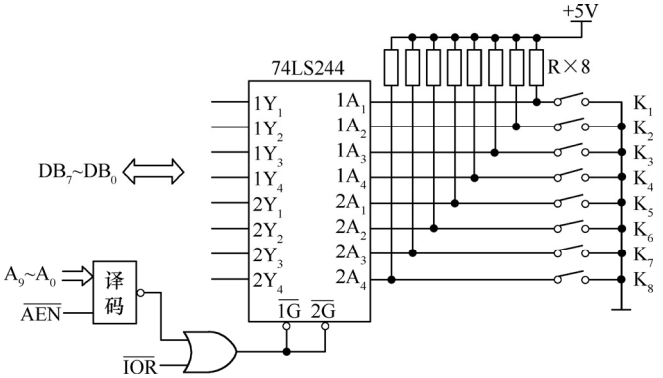


图 6-4　三态缓冲器构成输入接口

電子工業出版社

此接口电路只有数据端口，输入操作时，若开关的状态正发生变化，则输入的数据不可靠。

2．数据锁存器

数据总线是 CPU 和外部交换数据的公用通道，当 CPU 把数据送给输出设备时，只有执行总线周期的部分阶段总线会送出有效数据，因而必须利用数据锁存器及时把数据锁存起来，以便较慢的外设有足够的时间进行处理，使 CPU 和总线能够脱身去做其他工作。

74LS373 和 Intel 8282 都是常用的 8 位锁存器。在微机系统接口中，通常采用 8D 触发器 74LS273 进行数据锁存，其引脚及内部结构如图 6-5 所示，可以构成简单的输出接口，如图 6-6 所示。

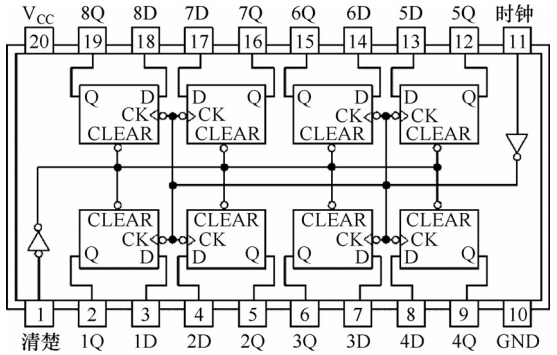


图 6-5　74LS273 引脚及内部结构

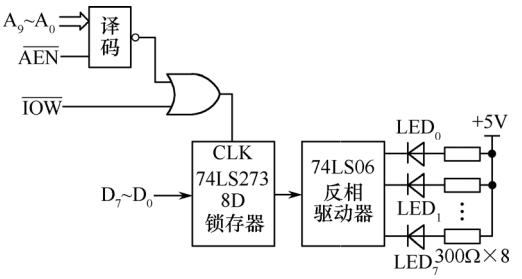


图 6-6　数据锁存器构成输出接口

当 CPU 对此端口进行输出操作时，地址译码和 $\overline{\text{IOW}}$ 同时有效。在 $\overline{\text{IOW}}$ 的后沿，将数据总线上的数据锁存到 74LS273 的输出端，再通过驱动电路控制发光二极管。此处取 $\overline{\text{IOW}}$ 的上升沿锁存是为了等待数据总线稳定。

6.2　输入/输出的传送方式

在微机系统中，CPU 与外设之间信息交换的输入/输出方式一般分为 3 种：① 程序控制的输入/输出；② 中断控制的输入/输出；③ 直接存储器存取（DMA）。它们各有优缺点，在实际使用时可根据具体情况进行选择。

6.2.1　程序控制的输入/输出

程序控制的输入/输出方式是指在程序中安排相应的 I/O 指令来控制输入/输出，完成与外设的信息交换。这种方式适用于预先知道何时需要进行数据传送的场合，根据需要将有关 I/O 指令插入到程序的相应位置。

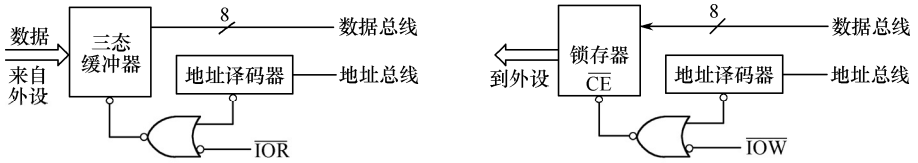
根据外设的不同性质，该传送方式又分为无条件传送及查询传送两种。

（1）无条件传送

有些设备的工作情况简单，状态很少发生变化。比如数码管只要 CPU 将数据的显示代码传送给它，就可立即显示相应数据。这时，CPU 不用查询外设的工作状态，默认外设始终处于准备好或空闲状态。在 CPU 认为需要时，随时与外设交换数据，这种传送方式就是

電子工業出版社

无条件传送方式。采用无条件传送方式工作的 I/O 接口电路如图 6-7 所示。



(a) 无条件传送的输入方式　　(b) 无条件传送的输出方式

图 6-7　无条件传送方式的 I/O 接口电路

【例 6-1】 采用无条件传送方式的 I/O 接口电路，用发光二极管显示开关的开合状态。

设计的接口电路如图 6-8 所示。程序中输入接口和输出接口使用相同地址，程序段如下：

```
NEXT:   MOV     DX, PORT_IN
        IN      AL, DX                  ; 通过输入接口读入开关状态
        NOT     AL
        OUT     DX, AL                  ; 通过输出接口控制发光二极管显示
        CALL    DELAY
        JMP     NEXT
```

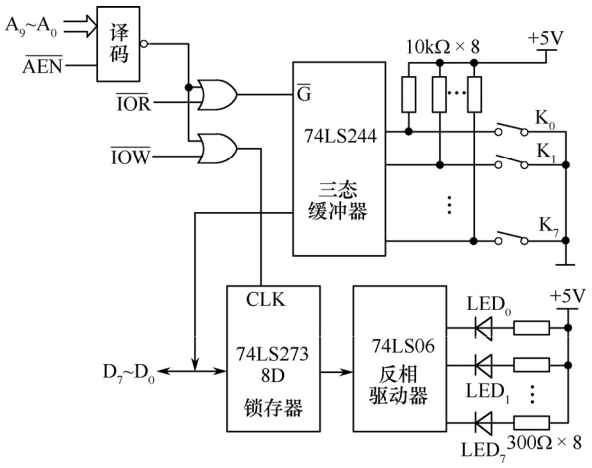


图 6-8　无条件传送举例

输入时，认为开关数据已送至三态缓冲器，CPU 执行相应的 IN 指令，地址信号经地址译码器译码后与 $\overline{\text{IOR}}$ 信号结合，选通数据输入端口，使开关数据经数据总线送往 CPU。这样做的条件是 CPU 在执行 IN 指令时，外设的数据是准备好的。

输出时，认为锁存器是空的，CPU 执行 OUT 指令，地址信号经地址译码器译码后与 $\overline{\text{IOW}}$ 信号结合，选通数据锁存器，使数据经数据总线送往锁存器，再由它送至 74LS06 驱动后加至发光二极管。这种情况下，CPU 在执行 OUT 指令时，外设是准备好的。

（2）查询传送

查询传送适用于 CPU 与外设异步工作的情况。在此方式中，CPU 首先查询外设的状态，只有在外设处于就绪状态时才与外设进行数据交换，否则一直处于查询等待状态。这时在查询方式的接口电路中，应同时含有数据和状态两种端口，以使 CPU 在与外设交换数据之前读取外设的状态信息。

電子工業出版社

① 查询式输入。一种采用查询式输入的接口电路如图 6-9 所示，地址译码电路输出低电平有效。一个数据输入端口用以读取外设的数据信息，一个状态输入端口用以读取外设的状态信息。设数据端口地址用符号 DATAS 表示，状态端口地址用符号 STATUS 表示。采用查询式输入的工作流程如图 6-10 所示。

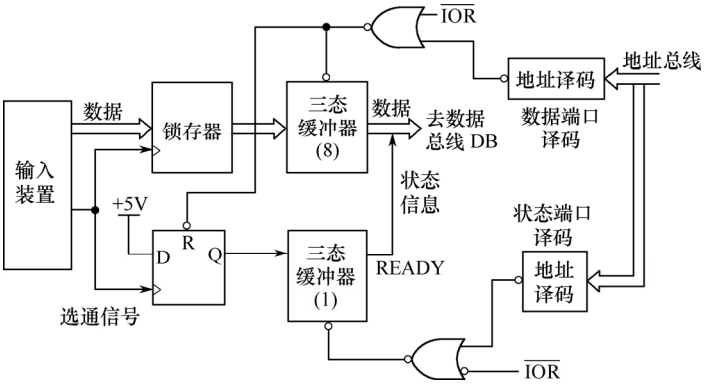


图 6-9　查询式输入的接口电路

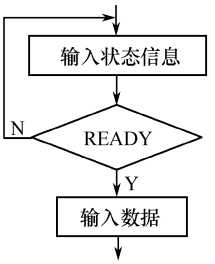


图 6-10　查询式输入的工作流程

当输入装置将数据准备好后，发出一个选通信号。此信号一边将数据送入数据锁存器，一边使 D 触发器置“1”（设“1”为“准备就绪”状态），发出 READY 状态信号。状态信号在这里只占用 1 位数据线，将其接至 D_7 位。CPU 先查询状态端口的 READY 信号（IN　AL, STATUS），当 READY 有效（D_7 位为“1”）时，才读取数据端口数据（IN　AL, DATAS）。同时，通过异步清“0”清除状态端口信息，表示数据已被取走，尚无准备好的数据可取。

【例 6-2】 结合图 6-9 的具体输入接口电路，编写如下查询式输入程序：

```
IN_TEST: IN      AL, STATUS          ; 读入状态信息
         TEST    AL, 80H             ; 检查 READY 是否为 1
         JZ      IN_TEST             ; 条件不满足，继续查询
         IN      AL, DATAS           ; 条件满足，读入数据
```

当系统中外设的状态信息不超过 8 位时，可以合并用同一个状态端口。

② 查询式输出。查询式输出也是先查询状态端口（IN　AL, STATUS），了解外设的工作状态，在其“空闲”时，再通过数据端口输出数据（OUT　DATAS, AL），否则继续查询。一种采用查询式输出的接口电路如图 6-11 所示，对应的工作流程如图 6-12 所示。

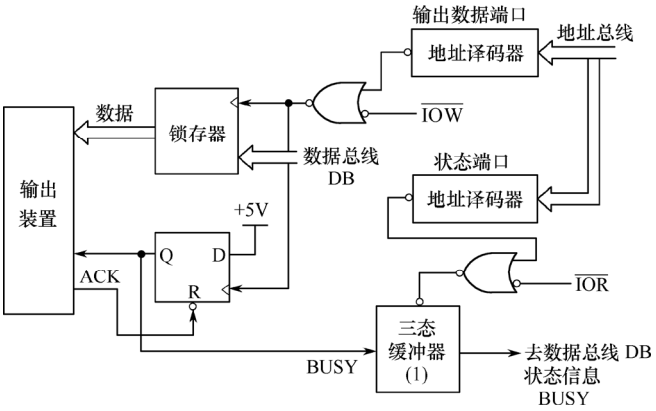


图 6-11　查询式输出的接口电路

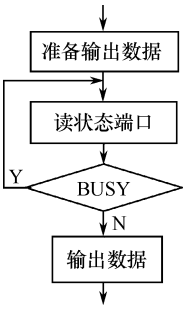


图 6-12　查询式输出工作的流程

電子工業出版社

CPU 先通过状态端口查询 BUSY（设为数据线的 D_7 位）是否空闲（设“0”表示空闲），空闲则通过数据端口输出数据，否则就一直查询 BUSY 的状态。CPU 查询 BUSY 为“0”后，向输出装置输出数据，同时置 D 触发器输出 BUSY 为“1”。在输出装置输出数据之前，BUSY 一直为“1”，以避免 CPU 写入新的数据；当输出装置输出数据后，回送一个 $\overline{\text{ACK}}$ 信号，使 D 触发器清“0”，表示再次进入空闲状态。

【例 6-3】 结合图 6-11 的具体输出接口电路，编写如下查询式输出程序：

```
            MOV     BX, OFFSET  STORE
OUT_TEST:   IN      AL, STATUS              ; 读入状态信息
            AND     AL, 80H                 ; 检查 BUSY 位
            JNZ     OUT_TEST                ; BUSY 则等待
            MOV     AL, [BX]                ; 空闲，则从缓冲区 STORE 中取数据
            OUT     DATAS, AL               ; 输出数据
            INC     BX
```

查询方式的优点是，硬件接口电路不太复杂，软件容易实现。但 CPU 需要花费大量的时间不断查询外设的工作状态，因而 CPU 的使用效率并不高。

此外，在利用查询方式进行 I/O 操作时，如果系统中有多个 I/O 设备需要查询，通常用轮流查询的方式来检测各外设的状态，先查询到的将先被处理。在实时控制系统中，若采用查询方式工作，有时会由于一个外设的输入/输出操作未处理完毕，就不能处理下一个外设，从而可能延误其他外设的数据传送，影响系统数据处理的实时性，甚至会由于某外设出现故障而一直无法就绪，从而导致查询无限循环（为避免陷入这种循环，实际程序中常加入超时判断等措施）。因此，查询式传送只适合 CPU 负担不重、要求服务的外设对象不多且任务相对简单的场合。

为了提高 CPU 的效率以及使系统具有更好的实时性能，通常采用中断传送方式。

6.2.2 中断控制的输入/输出

在 CPU 运行程序期间，遇到某些特殊情况（被内部或外部事件所打断），暂时中止原先程序的执行而转去执行一段特定的处理程序，这一过程就称为中断（Interrupt）。这段特定的处理程序称为中断服务程序。

中断控制的输入/输出方式，也称为中断传送方式，是指在外设就绪时主动向 CPU 发出中断请求，从而使 CPU 去执行相应的中断服务程序，完成与外设间的数据传送。因而该方式实时性好，在外设未准备就绪时，CPU 还可以处理其他事务，工作效率较高。采用中断方式输入数据的一种接口电路如图 6-13 所示。

当输入装置输入一个数据时，发出选通信号，该信号将数据存入锁存器，同时使 D 触发器置“1”，发出中断请求。若中断是开放的，则 CPU 接收中断请求信号并将当前指令执行完后，暂停正在执行的程序，发出中断响应信号 $\overline{\text{INTA}}$。这时外设将一个中断类型码放到数据总线，CPU 依据该中断类型码转入相应的中断服务程序。在中断服务程序中通过数据端口读取数据，同时清除中断请求标志。中断处理完毕，CPU 返回被中断的程序继续执行。

与查询方式数据传送相比，中断传送方式提高了 CPU 的工作效率，系统具有更好的实时性能。但由于中断请求的出现时刻具有随机性，因而何时执行中断服务程序事先无法预知，采用中断方式传送数据时程序设计应更为完善、周密。

電子工業出版社

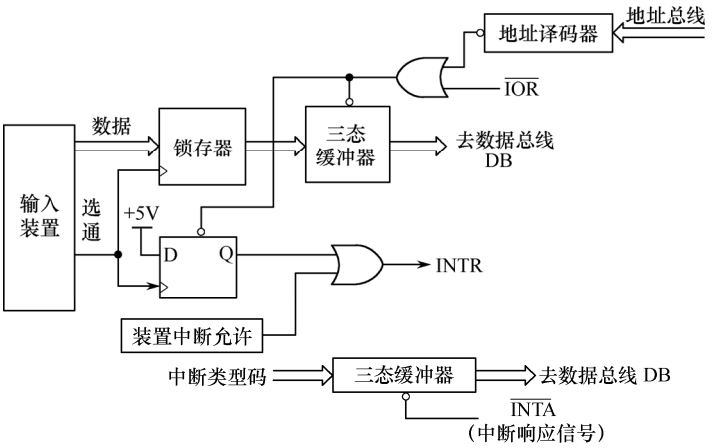


图 6-13　中断方式输入数据的一种接口电路

6.2.3　直接数据通道传送

中断传送方式虽可提高CPU的效率，但仍然是通过CPU执行一些指令来实现数据传送，这需要一定的时间，对于高速 I/O 设备以及成组交换数据的情况，速度太慢。所以在这种场合希望采用全硬件控制的方式，在外设与内存之间直接进行数据交换（Direct Memory Access，DMA），而不通过 CPU 执行指令进行。

DMA 数据传送方式在外设与内存之间建立了直接的数据通道，所有控制由 DMA 控制器（DMAC）来完成。通常，系统的数据、地址和控制总线是由 CPU 管理的。在 DMA 方式中，要求 CPU 让出这些总线，即要求 CPU 将与这些总线相连的引脚输出为高阻状态，而由 DMAC 来接管。

某输入设备使用 DMA 方式向存储器输入数据的示意如图 6-14 所示，数据传送的工作流程如图 6-15 所示。其工作过程如下：

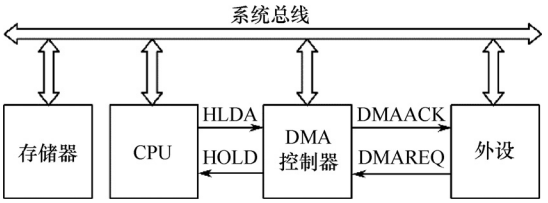


图 6-14　DMA 传送示意

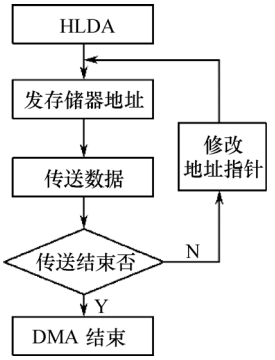


图 6-15　DMA 工作流程

① CPU 先把 DMAC 的工作方式、要写入的存储单元的首地址以及传送字节数等写到 DMAC 的内部寄存器中。

② 一旦输入设备有传送要求，CPU 将向 DMAC 发“DMA 请求”DMAREQ（该信号应维持到 DMAC 响应为止）。DMAC 收到请求后，向 CPU 发“总线请求”信号 HOLD，表示希望占用总线（该信号应在整个传送过程中维持有效）。CPU 在当前总线周期结束时响应

電子工業出版社

请求，向 DMAC 回“总线响应”信号 HLDA，表示已放弃总线。此时，DMAC 再向输入设备回送“DMA 响应”信号 DMAACK，该信号将清除 DMA 请求触发器，意味着传送开始。

③ 传送开始，DMAC 向输入设备送读控制信号 $\overline{\text{IOR}}$，同时向存储器送存储单元地址和写控制信号 $\overline{\text{MEMW}}$，于是完成 1 字节的传送。

④ DMAC 自动增减内部地址和计数，并据此判断任务是否完成，如果传送尚未完成，则重复上一步继续进行传送；如果传送完成，则将使发往 CPU 的“总线请求”信号 HOLD 无效，从而结束 DMA 传送，CPU 重新接管总线。

6.3 中断技术

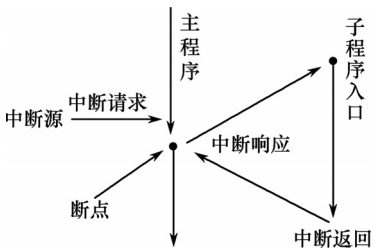


图 6-16 中断时的程序响应过程

中断处理是微处理器应具备的基本功能之一。当 CPU 响应中断，会暂停当前主程序，然后跳转执行中断服务，待中断服务执行完毕，再返回主程序的暂停位置（断点）继续执行。中断的程序响应过程如图 6-16 所示。

不同的微处理器中断响应时刻略有不同：CISC 微处理器是在一条指令执行结束后响应中断，而 RISC 则可以在一条指令执行的适当地方响应中断，与 CISC 相比，其中断响应更及时。

6.3.1 中断的基本概念

1．中断源

系统中引起中断的原因或能发出中断申请的来源，统称为中断源。中断源可以来自外部或内部，总结起来中断源有以下两种。

（1）外部中断源

外部中断源主要包括：① I/O 设备，如键盘、显示器（CRT）、打印机等；② 数据通道，如磁盘、采样电路及单片机中的串行口请求中断等；③ 实时钟，如外部所需的定时电路和单片机中的定时器/计数器等；④ 故障源，如掉电、存储器奇偶校验错等。

（2）内部中断源

内部中断源主要包括：① CPU 指令执行产生的异常，如被 0 除、溢出和单步执行等；② 执行软件中断指令，如 80x86 微机系统中 DOS 功能调用 INT 21H、BIOS 中断调用 INT 10H、ARM 处理器中的软中断异常指令 SWI、MIPS 处理器中的 SYSCALL 指令等。

2．中断处理过程

对于不同的微机系统，中断处理的具体过程不尽相同，即使是同一台微型计算机，由于中断方式的不同，中断处理也会有差别。但一个完整的中断处理的基本过程应包括：中断请求、中断判优、中断响应、中断处理和中断返回 5 个基本阶段。下面将以可屏蔽中断为例详细讨论中断的处理过程，如图 6-17 所示。

（1）中断请求

中断源产生中断请求的条件对不同的中断源是不一样的。如 I/O 设备申请中断，必须

電子工業出版社

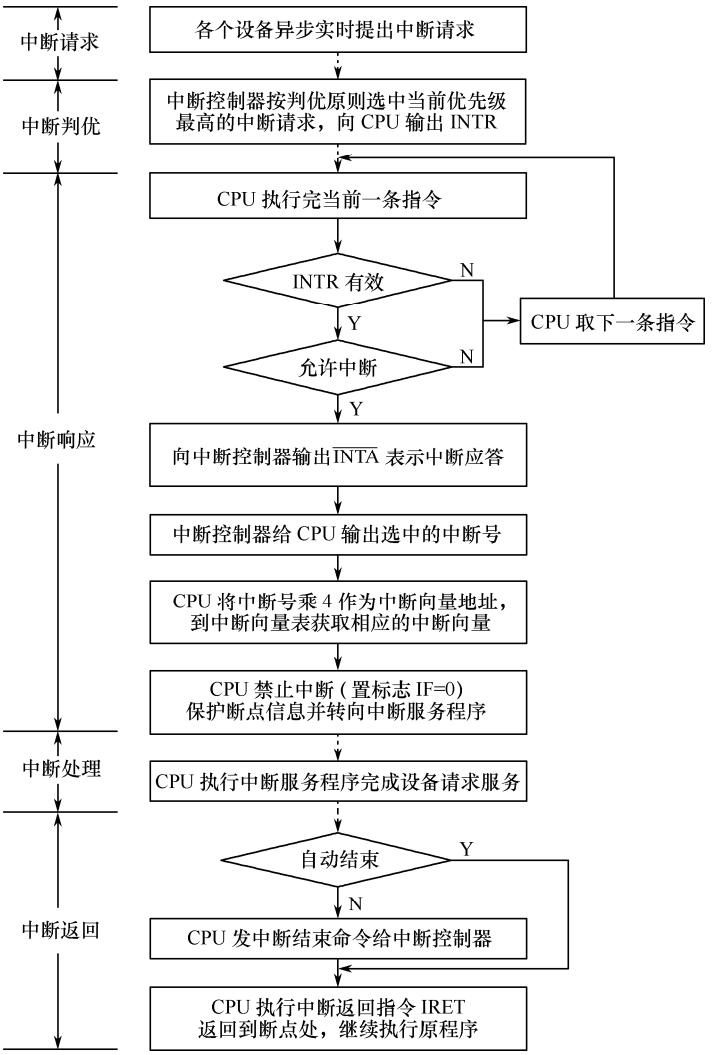


图 6-17　可屏蔽中断的处理过程

在 CPU 的相应引脚上产生中断申请信号，而除法出错中断请求的条件则是除数为零或商超过结果寄存器数的表示范围。中断源发出中断请求信号后，即进入中断判优阶段。

（2）中断判优

由于中断产生的随机性，有可能出现两个或两个以上的中断源同时提出中断请求的情况。这时就要根据中断源的轻重缓急，给每个中断源确定一个中断级别（中断优先权）。当多个中断源同时提出中断请求时，找出优先权级别最高的中断源响应它的请求；处理完毕再响应级别较低的中断源的请求。

中断判优的另一个作用是，决定是否能实现中断嵌套。如果 CPU 在响应某一中断请求时，出现了更高级别的中断源的请求，则中断判优电路允许新的中断源向 CPU 提出中断请求，从而暂时中止正在服务的当前中断服务程序，转去为高级中断源服务。等高级中断处理完毕，再返回到原中断服务程序断点处继续执行，这个过程称为中断嵌套。中断嵌套可以有多级，图 6-18 是多级中断嵌套的情况。

電子工業出版社

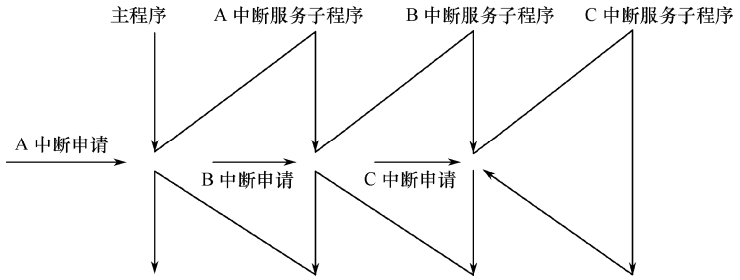


图 6-18　CPU 中断嵌套

（3）中断响应

对于可屏蔽中断，仅有中断请求还不一定能实现中断。除了优先权级别高低的条件，一般 CPU 包含中断控制寄存器，只有寄存器中的中断允许标志位允许的情况下，处理器才响应中断。如 80x86 的 IF 标志位，ARM 中的中断屏蔽寄存器，MIPS 处理器中的 SR 寄存器。

当 CPU 响应中断时，还会自动执行一些处理，如保护状态标志、保护断点和跳转到中断服务程序的入口。中断服务程序的入口地址通常被称为中断向量，这些中断向量统一存放在指定的内存空间中，该空间被称为中断向量表。在 80x86 中，中断向量表位于内存中最低的 1K 字节存储区域。

CPU 跳转到中断服务程序的入口后，即可执行相应的中断处理。

（4）中断处理

中断处理由中断服务程序完成。在中断服务程序中必须做以下工作。

① 保护现场。在执行中断服务程序时，需要先保护中断服务程序中要使用的寄存器内容，以防止影响当前程序的运行，中断返回前再将其恢复。中断服务程序一般使用堆栈操作指令完成现场保护，如 80x86 的 PUSH 指令。MIPS 处理器没有专门的堆栈操作指令，通过存储器访问指令 SW 和 LW 进行堆栈操作，来完成上述功能。

② 开中断。CPU 响应中断、保护状态标志后会自动关闭中断。若进入中断服务程序后允许中断嵌套，则需用指令开中断，如 80x86 的 STI 指令。

③ 执行中断服务处理程序。可根据不同的中断目的编写不同的中断服务处理程序。中断服务处理程序不宜过长或过于复杂，否则运行时既容易出乱，也影响其他中断源的及时处理。

④ 关中断。相应的中断服务处理结束后，为确保现场恢复不被打断，需使用指令关中断，如 80x86 的 CLI 指令。

⑤ 恢复现场。把先前保存的相关寄存器按入栈时相反的顺序弹出，使这些寄存器恢复到中断前的状态。

（5）中断返回

中断服务程序的最后是中断返回指令，执行返回指令后，CPU 会自动弹出断点信息，送给指令指针（如 IP 和 CS），并恢复标志寄存器的内容，以便回到断点处继续执行原程序。

6.3.2　中断优先权

通常，系统中多个中断源的请求都是送到 CPU 的同一引脚。因此对所有中断源，必须根据其中断的性质及处理的轻重缓急来安排优先权，一般采用软件排优和硬件排优两种

電子工業出版社

方法。

1．软件排优

软件排优是指各中断源的优先权主要由软件安排，与硬件电路关系不大。一种配合软件排优使用的电路如图 6-19 所示。若干外设的中断请求信号相“或”后，送至 CPU 的中断接收引脚（如 INTR）。这样只要任一个外设有中断请求（“1”状态），CPU 便可响应中断。在中断服务程序的开始处可安排一段优先权的查询程序，由 CPU 读取外设中断请求状态端口，然后根据预先确定的优先权级别逐位检测各外设的状态，若有中断请求，就转到相应的处理程序入口，其流程如图 6-20 所示。

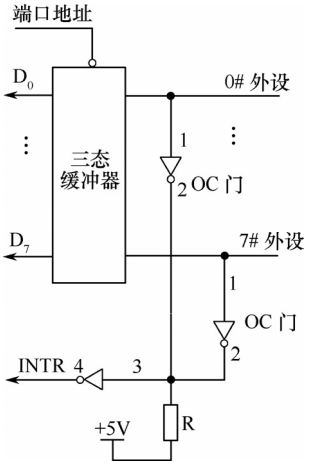


图 6-19　一种软件排优电路

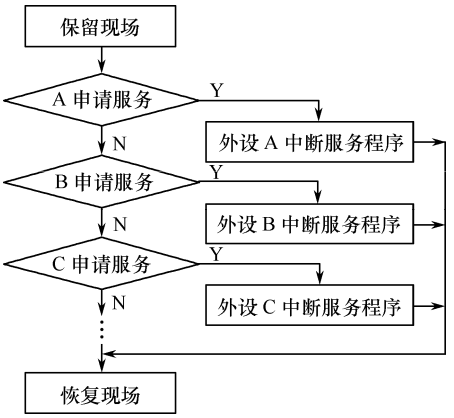


图 6-20　软件优先权查询流程

查询的顺序反映了各中断源的优先权的高低，最先查询的外设其优先级别最高。该方法的优点是，节省硬件，优先权安排灵活；缺点是，查询需要耗费时间，可能影响中断的实时性。

2．硬件排优

硬件排优是指利用专门的硬件电路或中断控制器对系统中各中断源的优先权进行安排。

（1）硬件排优电路

链式优先权排队电路是一种简单的中断优先权硬件排优电路，如图 6-21 所示。采用该方法时，每个外设对应的接口上连接一个逻辑电路，这些逻辑电路构成一个链，称为菊花链，由该菊花链来控制中断响应信号的通路。

当某个外设有中断请求时，CPU 如果允许中断，则发出 $\overline{\text{INTA}}$ 信号，如果链条前端的外设无中断请求信号，该级菊花链逻辑电路就会允许 $\overline{\text{INTA}}$ 自动地往后传递，一直传到发出中断请求的外设（该外设经过菊花链路得到的中断应答为“0”）。此时，该外设的菊花链逻辑电路对后面的菊花链逻辑电路实现阻塞，使 $\overline{\text{INTA}}$ 信号不再传到后面的外设（后面的外设得到的中断应答始终为“1”）。因而，采用菊花链电路的各外设的中断优先权由其在链中的位置决定，处于链条前端的比处于链条后端的优先权高。当某外设收到 $\overline{\text{INTA}}$ 后，可向 CPU 送出本身的中断类型码。

若 CPU 正执行某个中断服务程序时，又有级别较高的外设提出中断请求，由于菊花链电路中级别低的外设不能封锁级别高的外设得到中断响应信号，故仍可响应该中断请求，从而发生中断嵌套。

電子工業出版社

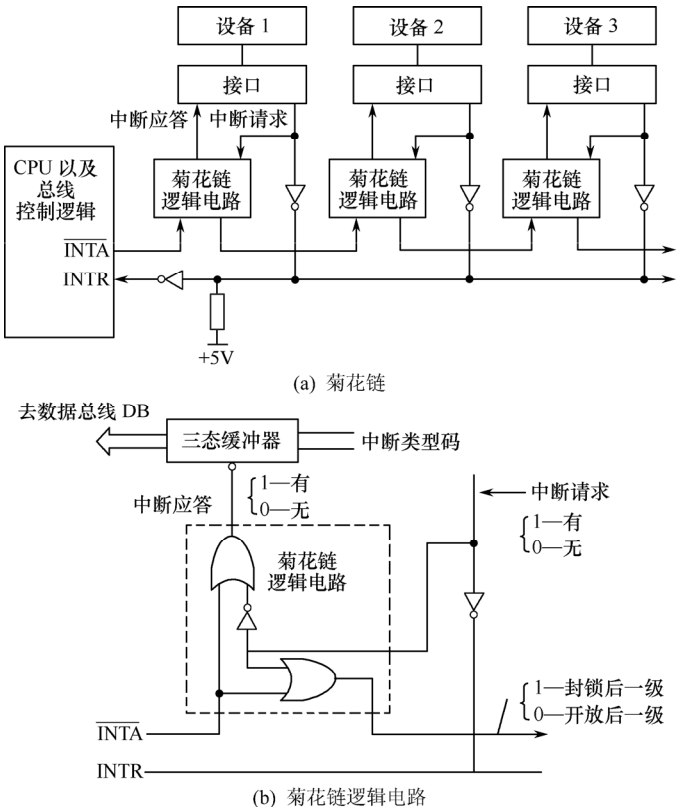


图 6-21　链式优先权排队电路

（2）可编程中断控制器

可编程中断控制器是当前微机系统中解决中断优先权管理的常用办法。通常，中断控制器包括：中断优先权管理电路、中断请求锁存器、中断服务寄存器和中断屏蔽寄存器等。中断控制器也可认为是一种接口，外设提出的中断请求经该环节处理后，再决定是否向 CPU 传送，CPU 接受中断请求后的中断响应信号也是送给该环节处理的，以便得到相应的中断类型码。有关中断控制器的详细说明，将在 6.5 节和 6.6 节介绍。

6.4　80x86/Pentium 中断系统

6.4.1　中断结构

Intel 80x86 系列微机的中断系统可以处理 256 个中断源，每个中断源都有对应的中断类型码（0～255）供 CPU 识别。在 8086/8088 系统中，中断源分为外部（硬件）中断与内部（软件）中断两类，如图 6-22 所示。

1．外部中断

外部中断是由外部硬件产生的突发事件，包括非屏蔽中断和可屏蔽中断两种，通过 CPU 的 NMI 引脚或 INTR 引脚向 CPU 申请中断。外部中断的发生具有随机性，可能由 I/O 请求引起，或由时钟或硬件故障引起。

電子工業出版社

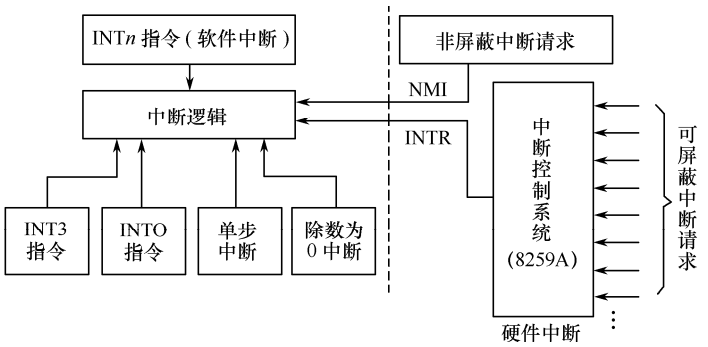


图 6-22　8086/8088 中断分类

（1）非屏蔽中断

由 NMI 引脚引起的中断称为非屏蔽中断，不受中断允许标志位 IF 的限制，不能在 CPU 的内部被屏蔽掉，中断类型码固定为 2。NMI 信号是上升沿触发，CPU 内部会把该信号锁存起来，但要求该信号的有效高电平持续 2 个时钟周期以上。CPU 接收到非屏蔽中断请求信号后，不论当前正在做什么，都会在执行当前指令后立即响应中断请求而进入相应的中断处理。在实际系统中，非屏蔽中断通常用来处理系统中出现的重大故障或紧急情况，如系统掉电处理、紧急停机处理等。与一般硬件中断不同，对于 NMI，CPU 不执行中断响应周期。

（2）可屏蔽中断

一般外部设备提出的中断请求是从 CPU 的 INTR 端引入的，所产生的中断称为可屏蔽中断。可屏蔽中断受中断标志位 IF 的影响，只有在 IF=1 时，CPU 才会在执行当前指令后响应可屏蔽中断请求，所以 INTR 信号有效也必须保持到当前指令执行结束。如果 IF=0，中断被屏蔽，即使有中断请求 CPU 也不会响应。在 IBM PC 系列微机中，通常外部设备提出的中断请求首先由中断控制器 8259A 预处理，再决定是否向 CPU 的 INTR 引脚提出中断请求。CPU 响应可屏蔽中断请求信号后，会通过 $\overline{\text{INTA}}$ 引脚送出两个负脉冲，并获得中断源对应的中断类型码。

2．内部中断

内部中断是 CPU 执行某些指令或指令执行的结果使标志寄存器中某个标志位置 1 而引发的，与外部硬件电路完全无关，中断类型码或由系统预先设定，或包含在指令中。

① 除法出错中断。当 CPU 执行除法指令时，若发现除数为 0 或商超过了结果寄存器所能表示的数的范围，则立即产生一个中断类型码为 0 的除法出错中断。该中断的服务处理一般由操作系统安排。

② 单步调试中断。若标志寄存器中 TF=1，CPU 便处于单步工作方式，即每执行完一条指令就自动产生一个中断类型码为 1 的中断。单步方式为系统提供了一种方便的调试手段，成为能够逐条指令地观察系统操作的一个“窗口”。如 DEBUG 中的跟踪命令 T，就是将 TF 置 1，进而去执行一个单步中断服务程序，以跟踪程序的具体执行过程，找出程序的问题所在。

说明，所有类型的中断在其处理过程中，CPU 会自动地把标志压入堆栈，然后清除 TF 和 IF 位。因此当 CPU 进入单步处理程序时，就不再处于单步工作方式，而以正常方式工作。只有在单步处理结束，从堆栈中弹出原来的标志后，才使 CPU 又返回到单步方式。注意，80x86 指令系统中没有设置或清除 TF 标志的专门指令，可利用 PUSHF 和 POPF 来实现对

電子工業出版社

TF 的置位和复位，程序段如下：

```
PUSHF                    ; 标志寄存器 FLAGS 入栈
POP     AX               ; (FLAGS)→AX
OR      AX, 0100H        ; TF 标志置 1，若要使 TF 清 0，应改为 AND  AX, 0FEFFH
PUSH    AX
POPF                     ; 改变标志寄存器的值
```

③ INTO 指令，为溢出中断指令。若算法操作结果产生溢出（OF=1）时，执行 INTO 指令后立即产生一个中断类型码为 4 的中断。INTO 为程序员提供了一种处理算术运算出现溢出的手段，通常与算术指令配合使用。

④ INT　*n* 指令。INT *n* 是双字节代码指令，第一字节为操作码，第二字节为中断类型码 *n*（*n*=0～255）。CPU 执行一条这种指令便发生一次中断。

其中，INT　3 是 INT　*n* 的特例，是单字节指令，因而能被方便地插入到程序的任何地方，专供调试程序时在程序中设置断点之用。插入 INT　3 指令之处便是断点，在其中断服务程序中可显示有关的寄存器和存储单元等内容，以分析到断点为止程序运行是否正确。

80386 及以上 CPU 扩展了一些内部中断，把许多执行指令过程中产生的错误也纳入中断处理的范围，并将它们和内部中断一起统称为异常中断，简称异常；将通常意义上的外部中断简称为中断。因此，对 80386 及以上 CPU 来说，也可将中断源分为中断和异常两类。

异常包括由指令引起的异常（如 INT　*n*、INTO、BOUND 等）和处理器检测异常（如除法出错、无效操作码、堆栈故障、段/页不存在、浮点协处理器、单步调试异常等）。所有的异常按其发生异常的报告方式和异常服务程序返回方式的不同，又可归为 3 类：失效、陷阱和终止。

① 失效一般在引起失效的指令启动之后、执行之前被检测到，且在处理异常的中断服务完成后返回该条指令，重新启动并执行完毕。如在读虚拟存储器时，当访问的页或段不在物理存储器中，便产生一个失效字，引起异常中断；其中断服务程序立即从磁盘上读取这个页或段的内容到物理内存中，再返回主程序中重新执行这条指令。

② 陷阱是在产生陷阱的指令执行完后才被报告，且其中断服务程序结束后返回到主程序中该条指令的下一条指令。如中断指令 INT　*n* 属于此类异常。

③ 终止则是一种对引起异常的指令的确切位置无法确定的异常，如硬件错误或系统表中的非法数值等造成的异常。当出现此类异常时，原来的程序已无法继续执行，只好终止，而由中断服务程序重新启动操作系统并重建系统表格。

6.4.2　中断向量表

在 80x86/Pentium 中断系统中，每个中断源对应一个确定的 8 位中断类型码。CPU 在响应中断后，会根据中断类型码查询中断向量表转入相应的中断服务程序。

中断向量表是指内存最低 1 KB（00000H～003FFH）的存储区域，其中存放对应 256 个中断源的中断服务程序的入口地址（中断向量）。每个中断向量占 4 字节，前 2 字节为中断服务程序入口地址的偏移地址（IP），后 2 字节为段地址（CS）。中断向量按照中断类型码的序号，在中断向量表中有规则地顺序存放，如表 6-3 所示。

中断类型码与中断向量在向量表中所在位置（中断向量地址指针）之间的对应关系为：

中断向量地址指针 ＝ 中断类型码×4

電子工業出版社

在表 6-3 中，前 5 个中断类型是 80x86 规定的专用中断，有固定的含义和处理功能；类型码 5～31 是 Intel 公司为系统功能或将来扩展特殊用途而保留的；用户可以自行使用类型码 32～255。在一种具体的微机系统中，对用户可使用的中断类型码又有特殊的规定。如在 PC/AT 机中，类型码 70H～77H 已有安排，所以在系统开发和应用时应加以注意。

由于 ROM BIOS 和 MS-DOS 的版本不同以及多种 PC 主板，中断类型码的分配在不同的计算机上会有些差异。

表 6-3　中断向量表

分类	中断源	入口地址
专用中断	000H	类型 0 中断入口地址（除法出错）
	004H	类型 1 中断入口地址（单步中断）
	008H	类型 2 中断入口地址（NMI）
	00CH	类型 3 中断入口地址（断点中断）
	010H	类型 4 中断入口地址（溢出中断）
系统备用	014H	类型 5 中断入口地址
	⋮	⋮
	07CH	类型 31 中断入口地址
供用户使用	080H	类型 32 中断入口地址
	⋮	⋮
	3FCH	类型 255 中断入口地址

6.4.3　中断响应过程

80x86/Pentium 系统中各种中断的响应和处理过程略有不同，主要在于如何获取相应的中断类型码。

1．内部中断响应过程

内部中断包括系统使用的专用中断与用户使用的 INT *n* 指令。内部中断没有随机性，不受中断允许标志位 IF 的影响，而且其中除单步中断外都无法用软件禁止。当内部中断发生时，CPU 无须执行中断响应周期去获得中断类型码。对于专用中断，中断类型码是自动形成的；而对于 INT *n* 指令，类型码即为指令中给定的 *n* 值。

CPU 取得中断类型码后的处理过程如下：

（1）将中断类型码×4，作为中断向量表的地址指针；

（2）将 CPU 的标志寄存器入栈保护，此操作类似于 PUSHF 指令；

（3）清除 IF 和 TF 标志位，屏蔽新的 INTR 中断和单步中断；

（4）保存断点，即把断点处的 IP 和 CS 值压入堆栈，先压入 CS 值，再压入 IP 值；

（5）从中断向量表中取中断服务程序的入口地址，分别送至 CS 和 IP 中；

（6）按新的 CS:IP 执行中断服务程序。

在中断服务程序中，通常要保护现场，进行相应的中断处理，恢复现场，最后执行中断返回指令 IRET。IRET 的执行将使 CPU 依次恢复断点处的 IP 和 CS 值，恢复标志寄存器（相当于 POPF），于是程序就恢复到断点处继续执行。

2．外部中断响应过程

（1）非屏蔽中断响应。CPU 采样到非屏蔽中断请求时，自动提供中断类型码 2，其后的处理过程和内部中断一样。

（2）可屏蔽中断。当 INTR 信号有效时，如果中断允许标志 IF=1，则 CPU 就会在当前指令执行完毕后，响应外部中断请求，转入中断响应周期。中断响应周期有 2 个，每个响应周期都由 4 个 T 状态组成。在第一个中断响应周期，CPU 输出 $\overline{\text{INTA}}$ 负脉冲，表示 CPU 响应外设中断请求，并通知外设现在应该准备好中断类型码，同时用以撤销外设提出的 INTR 信号；在第二个中断响应周期，CPU 又输出 $\overline{\text{INTA}}$ 负脉冲，要求请求中断的外设在

電子工業出版社

收到第二个负脉冲以后（第二个中断响应周期的 T_3 状态前），把中断类型码通过数据总线送给 CPU（通常通过 8259A 传送），CPU 在 T_4 状态的前沿采样数据总线，获取中断类型码，如图 6-23 所示。

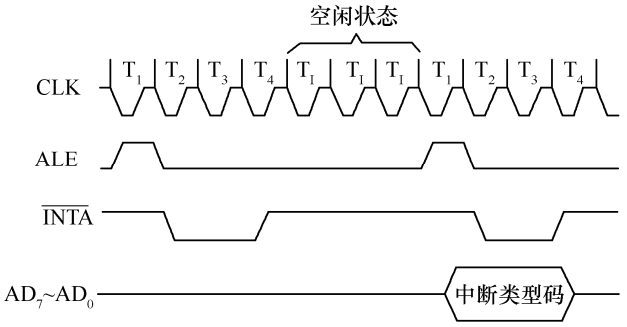


图 6-23　中断响应总线周期

CPU 获取中断类型码后的处理过程和内部中断一样，自动从中断向量表中取出中断服务程序的入口地址并转入中断服务。但使用过程中应注意两点：

① CPU 复位后，中断允许位处于关闭状态。因此要使中断得到响应，必须使用指令开中断。

② CPU 响应中断后，自动关中断。因此，要允许中断嵌套，必须在进入中断服务程序后使用指令重新开中断。

一个可屏蔽中断被响应时 CPU 实际执行的总线周期有 7 个：

① 执行两个中断响应总线周期（8086 在两个周期之间用 2～3 个空闲状态隔开），CPU 获得相应的中断类型码，将它左移两位形成中断向量表指针，存入暂存器。

② 执行一个写总线周期，把标志寄存器的内容压入堆栈。同时置 IF 和 TF 为 0，以禁止中断响应过程中其他可屏蔽中断的进入，同时也禁止中断处理过程中出现单步中断。

③ 执行两个写总线周期，把断点地址（当前的 CS:IP）的内容压入堆栈。

④ 执行两个读总线周期，从中断向量表中分别取出中断服务程序入口地址的偏移量与段基值送到 IP 和 CS 寄存器中。

3．中断响应及处理过程举例

【例 6-4】 设某输入设备在数据准备就绪后向 CPU 申请可屏蔽中断，中断类型码为 80H，中断响应后执行的中断服务程序名为 INTPROC。

在允许中断前，首先要写中断向量到中断向量表，即先进行中断初始化，程序段如下：

```
CLI
PUSHDS
MOV     AX，0000H                ；中断向量指针的段基址
MOV     DS，AX
MOV     BX，0200H                ；中断向量指针的偏移量
MOV     AX，OFFSET  INTPROC
MOV     [BX]，AX                 ；中断向量的偏移量部分 →中断向量表
MOV     AX，SEG  INTPROC
MOV     [BX+2]，AX               ；中断向量的段基址部分 →中断向量表
POP     DS
```


電子工業出版社

```
        STI
```

开中断后，执行任何指令期间都有可能接收到中断申请。假设执行下列程序中的 ADD 指令时收到中断申请，且 IF=1，则 CPU 响应中断，响应过程如图 6-24 所示。

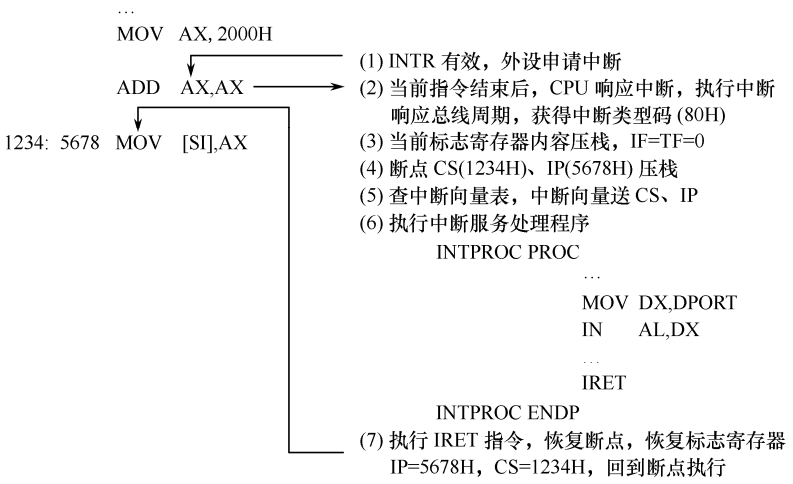


图 6-24　例 6-4 中断过程举例

4．各类中断的优先权

80x86/Pentium 系统中所有的中断都是在当前指令结束后处理的。在当前指令执行完毕，CPU 总是按照中断优先级别，从高到低自动查询中断类型，先查询到的中断先被响应。80x86/Pentium 系统的中断优先级别从高到低排列顺序为：内部中断（除法出错、INTO、INT *n*）、NMI、INTR、单步中断。

6.4.4　80386/80486/Pentium CPU 中断系统

80386/80486/Pentium CPU 在中断处理机制上有实模式和保护模式两种不同情况。

1．实模式下的中断与异常

80386 及以上 CPU 工作于实模式时，中断处理机制与 8086/8088 的基本一致，都可定义 256 种中断类型，并将中断向量表存放在系统物理存储器的最低 1 KB 区域。中断类型及各类中断处理流程也基本相同，都是通过中断类型码查找中断服务程序入口地址。表 6-4 给出了 8086～Pentium CPU 在部分微机中的中断分配表。

Intel 公司保留前 32 个中断类型为 Intel 各种 CPU 系列成员专用，最后 224 个可作为用户中断类型。表 6-4 中前 5 个中断类型在 8086～Pentium 的所有 Intel 系列 CPU 中都是相同的。其他中断类型存在于 80286 及向上兼容的 386、486、Pentium CPU 中，但不向下兼容 8086 或 8088。所以对于 80386～Pentium 微机系统，类型码 08H～0FH 既被内部中断使用，又供外部可屏蔽中断 IRQ_0～IRQ_7 使用，类型码 70H～77H 同样可同时供内部中断及外部可屏蔽中断 IRQ_8～IRQ_{15} 使用。为区分某个类型码是内部中断，还是由 INTR 引脚引起的外部中断，在处理之前，要判断出错代码中的 EXT 位，如果为“1”，则执行外部中断处理过程，否则执行内部异常处理过程。

電子工業出版社

表 6-4　8086/Pentium CPU 在部分微机中的中断分配表

类型码	8086/8088	80386/Pentium	
00H	除法出错	除法出错	
01H	单步中断	单步中断	
02H	NMI 中断	NMI 中断	
03H	断点中断	断点中断	
04H	溢出中断	溢出中断	
05H	屏幕打印	数组边界检查	
06H	保留	无效操作码	
07H	保留	协处理器无效	
08H	定时器时钟中断（IRQ_0）	检测到双精度异常	IRQ_0
09H	键盘中断（IRQ_1）	协处理器段超界（保留）	IRQ_1
0AH	8259A 级联（IRQ_2）	非法的任务状态段	IRQ_2
0BH	串行口 2（IRQ_3）	段不存在	IRQ_3
0CH	串行口 1（IRQ_4）	堆栈失效	IRQ_4
0DH	并行口 2（IRQ_5）	保护错误	IRQ_5
0EH	软盘中断（IRQ_6）	页失效	IRQ_6
0FH	并行口 1（IRQ_7）	保留	IRQ_7
10H	显示 I/O 功能程序	协处理器异常（含 80286）	
11H	设备配置检测	对齐检查（仅 80486SX）	
12H	存储器容量检测	机器检测（Pentium 系列）	
13H	磁盘 I/O 功能程序	磁盘 I/O 功能程序	
14H	串行通信 I/O 功能程序	串行通信 I/O 功能程序	
15H	盒式磁带机 I/O 功能程序	盒式磁带机 I/O 功能程序	
16H	键盘 I/O 功能程序	键盘 I/O 功能程序	
17H	打印机 I/O 功能程序	打印机 I/O 功能程序	
18H	ROM BASIC 入口	保留	
19H	引导程序入口	引导程序入口	
1AH	日时钟 I/O 功能程序	日时钟 I/O 功能程序	
1BH	键盘中止控制	键盘中止控制	
1CH	定时器报时	定时器报时	
1DH	显示器初始化参数设置	显示器初始化参数设置	
1EH	磁盘参数设置	磁盘参数设置	
1FH	图形字符集	图形字符集	
20H～3FH	DOS 中断	DOS 中断	
40H～6FH	保留，用户使用或未使用	保留，用户使用或未使用	
70H	实时时钟中断（IRQ_8）	保留	IRQ_8
71H	软件使其重新指向（IRQ_2）（IRQ_9）		IRQ_9
72H～74H	保留（IRQ_{10}～IRQ_{12}）		IRQ_{10}～IRQ_{12}
75H	协处理机中断（IRQ_{13}）		IRQ_{13}
76H	硬盘中断（IRQ_{14}）	保留	IRQ_{14}
77H	保留（IRQ_{15}）		IRQ_{15}
78H～FFH	未使用或保留给 BASIC 使用	保留	

電子工業出版社

2．保护模式下的中断与异常

保护模式下的中断处理过程与实模式相比有三点不同：① CPU 根据中断类型码从中断描述符表 IDT 而不是中断向量表获取中断服务程序的入口信息。② 中断过程中要对被中断的程序代码进行保护，即要进行特权级检查。③ 若有出错码，还要将出错码压入堆栈。

（1）中断描述符表

IDT 的起始位置通过中断描述符表寄存器 IDTR 设置，可置于内存的任意区域。在整个系统中，IDT 只有一个。它的作用同样是通过类型码引导程序转移到中断或异常处理程序中。但是其表项内容与实模式完全不同，IDT 包含的是门描述符而不是中断向量。门描述符可以是中断门、陷阱门或任务门。IDT 最多允许有 256 个门描述符，标识为门 0～255，对应 256 种中断类型。

（2）获得中断服务程序地址

使用中断门/陷阱门描述符进入中断服务程序入口的过程如图 6-25 所示。

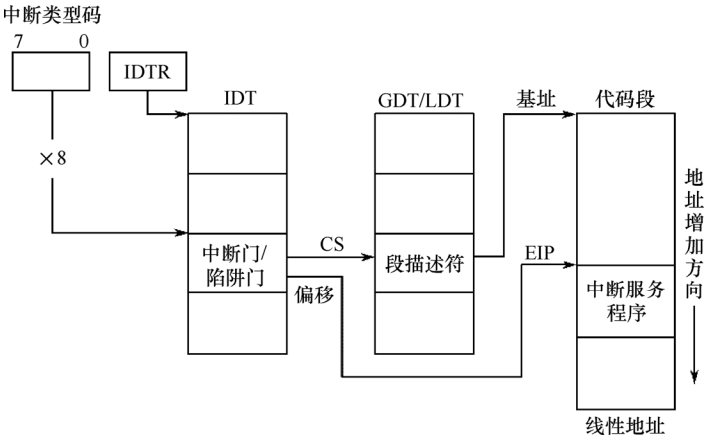


图 6-25　使用中断门/陷阱门进入中断程序入口的过程

具体描述如下：

① 将中断类型码 *n* 乘以 8 得到相应的中断门或陷阱门描述符首字节在 IDT 中的地址。

② 从 IDT 读出门描述符（8 字节），将其中的 16 位选择符装入代码段寄存器 CS。

③ 用 CS 寄存器从 GDT 或 LDT 中选择相应的段描述符。

④ 将段描述符中的基址（32 位）与门描述符中的偏移量（32 位）相加得到中断服务程序的入口地址（线性地址）。

中断门和陷阱门只能设置在 IDT 内。它们的不同之处在于，通过中断门访问中断服务程序将清除 EFLAGS 中的 IF 标志，而陷阱门则不改变 IF 内容。一般来讲，外部硬件中断由中断门描述符来配置。这样，在执行中断服务程序中，除非软件允许再次中断，否则可以防止另外的中断影响当前中断的处理过程。软件中断及内部中断一般用陷阱门。

（3）中断响应及异常处理。保护模式下的中断响应过程基本与实地址方式相似。响应中断时，要将标志寄存器 EFLAGS、段寄存器 CS、指令指针 EIP 的内容压入堆栈；清除 NT、TF、IF 标志（陷阱门不清除 IF）；如果异常导致错误的代码要保存，还需将错误码压入堆栈。与实模式下中断处理不同之处有以下两点：

① 需要通过类型码判断所使用的门描述符是否超出 IDT 的界限。若超出，则引起一般

電子工業出版社

保护异常错误。

② 从 IDT 中取得对应的门描述符后要进行相应的检查。

门描述符只能是任务门、286 中断门、286 陷阱门、386 中断门和 386 陷阱门，否则同样会引起一般保护异常错误。

门描述符中的 P 位必须是 1，表示该描述符可用，否则就引起段不存在故障。

中断时，中断服务程序的代码段必须比被中断程序的特权级高或相同（DPL≤CPL），从而对被中断的程序代码进行保护，否则将产生一般保护异常错误。如果 DPL=CPL，中断服务程序使用当前的堆栈；如果 DPL<CPL，将出现堆栈转换，主程序的堆栈指针（由 SS:ESP 提供）也将压入堆栈，而中断服务程序使用的选择符以及堆栈指针从当前所执行任务的 TSS 获得。CPU 自动将主程序堆栈中的 SS、ESP、FLAGS、CS、EIP 和出错码信息复制到中断服务程序的堆栈中。堆栈的变化过程如图 6-26 所示。

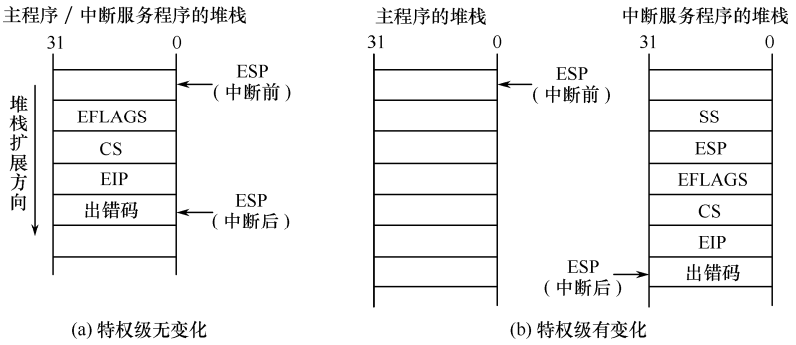


图 6-26　中断响应时堆栈的变化情况

③ 在保护模式下，除了可通过中断门或陷阱门实现由当前任务进行中断处理，也允许通过任务门实现由另一个任务完成中断处理。使用任务门的优点是，被中断程序的环境资源入栈和中断任务环境资源的加载全部自动进行，而且中断任务与其他任务隔离。缺点是，由于要保存大量的机器状态，处理程序较复杂，处理时间比中断门要长。

从异常或中断服务程序返回时，必须使用 IRET/IRETD 指令，两种指令分别应用于 16 位和 32 位操作。

6.5　8259A 可编程中断控制器

中断控制器用在多中断源微机系统中管理中断，实现对各中断源的优先权和中断类型码等进行安排；对外设送来的中断请求进行判优和中断屏蔽；在中断响应周期送出中断类型码等。

早期的 PC/XT 中使用的可编程中断控制器（PIC，Programmable Interrupt Controller）一般为 Intel 8259 系列产品，这种 PIC 只能够支持 8 个中断优先级，但是可以通过级联最多支持 64 个优先级。随着 PC 系统集成度的提高，从 80386 微机系统开始，中断控制器被包含在集成控制芯片中，典型的集成控制芯片是 82380。82380 芯片集成了多个不同功能的接口组件，其中 PIC 由三个增强功能的 8259A 中断控制器组成，可提供 15 个外部和 5 个内部中断请求输入，每个外部请求输入可以再级联一个 8259A 从控制器。为了与 8086/8088 兼容，其中断处理过程及编程和 8259A 基本保持一致。发展到 Pentium 系列微机系统，PIC 技术已

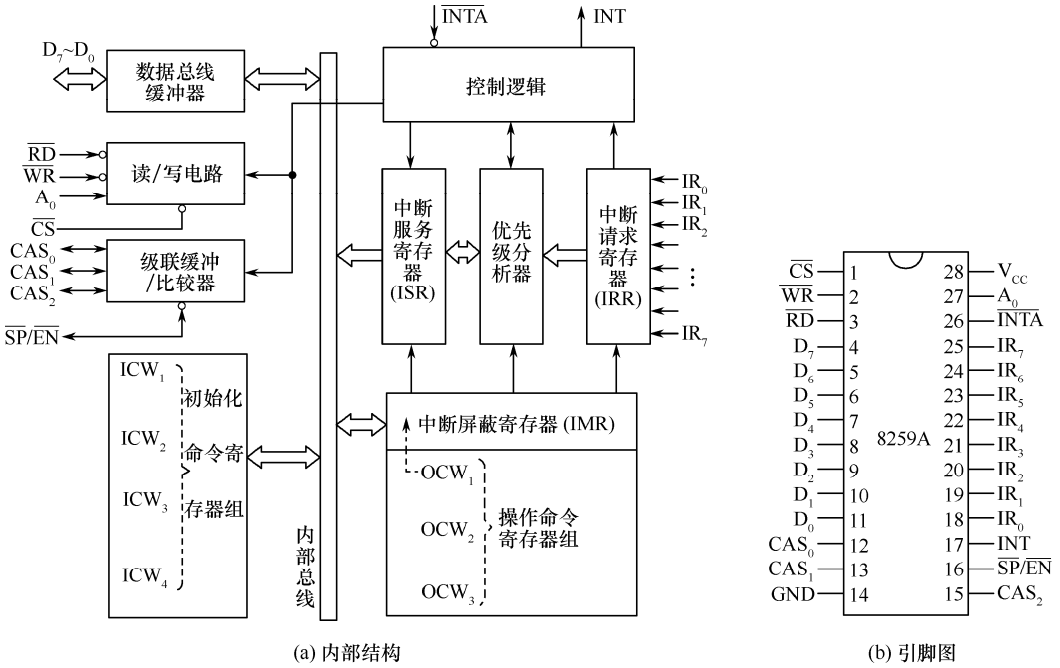
電子工業出版社

经被高级可编程中断控制器（APIC，Advanced Programmable Interrupt Controller）技术所取代。下面将以 Intel 8259A 为例展开介绍。

Intel 8259A 是一种可编程中断控制器，它是为 80x86 CPU 管理可屏蔽中断而设计的。一片 8259A 可以支持和管理 8 级优先权中断，多片级联最多可扩展至 64 级优先中断控制系统。8259A 有多种工作方式，包括中断请求触发方式、中断屏蔽功能及方式、中断优先级算法及方式、中断结束方式等，都可以通过编程来选择，以适应不同的应用场合。

6.5.1 8259A 芯片的内部结构与引脚

8259A 是有 28 个引脚的双列直插式芯片，内部结构与芯片引脚如图 6-27 所示。它由中断请求寄存器 IRR（Interrupt Request Register）、优先级分析器（即优先权判决电路）、中断服务寄存器 ISR（In-Service Register）、中断屏蔽寄存器 IMR（Interrupt Mask Register）、数据总线缓冲器、读/写电路、控制逻辑、级联缓冲/比较器、初始化命令与操作命令寄存器组等构成。



(a) 内部结构　　(b) 引脚图

图 6-27　8259A 芯片内部结构与芯片引脚图

IR_7～IR_0 是 8 条中断请求线输入线，连接 8 个 I/O 设备的中断请求信号。IRR 寄存所有要求服务的中断请求，当 IR_7～IR_0 中有中断请求时，IRR 中相应位就置 1。

优先权判决电路，它对 IRR 中的各中断请求进行分析，确定当前最高优先权的中断源，并在 CPU 响应中断请求的第一个响应周期将它选通送至 ISR 的相应位。ISR 寄存器中用 1 表征正在被服务的中断源。

IMR 存放是否屏蔽 IR_7～IR_0 各中断源的屏蔽码，其每一位对应一个中断级，为 1 时屏蔽该级中断，否则开放该级中断。

控制逻辑根据 CPU 对 8259A 编程设定的工作方式产生内部控制信号，并在适当时候对

電子工業出版社

CPU 发出中断请求信号 INT 和接收来自 CPU 的中断响应信号 $\overline{\text{INTA}}$，将中断响应脉冲 $\overline{\text{INTA}}$ 转换为各种内部控制信号。

读/写电路接收 CPU 的读/写命令。可以将来自 CPU 的初始化命令字（ICW）和操作命令字（OCW）写入相应的命令寄存器组，以规定 8259A 的工作方式和控制模式；也可通过它将这些寄存器组内容读出，以了解 8259A 芯片的内部状态信息。与读/写电路有关的引脚功能如表 6-5 所示。

表 6-5　与读/写电路有关的引脚功能

符号	名　　称	功 能 说 明
$\overline{\text{CS}}$	片选线	$\overline{\text{CS}}$ =0，芯片被选中，允许 CPU 读/写。一般由高位地址线译码得到
$\overline{\text{WR}}$	写命令信号线	$\overline{\text{WR}}$ =0，允许 CPU 把命令字（ICW 和 OCW）写入相应命令寄存器
$\overline{\text{RD}}$	读命令信号线	$\overline{\text{RD}}$ =0，允许 CPU 读取 IRR、ISR、IMR 三个寄存器的内容
A_0	端口选择线	用于片内寄存器选择。一般可直接接至地址总线的 A_0 位或其他位

数据总线缓冲器是 8 位双向三态缓冲器，用于连接系统数据总线和 8259A 芯片内部总线，以便编程时 CPU 对 8259A 写入控制字或读取状态字。

级联缓冲/比较器用于多片 8259A 之间的连接，使得中断源可由 8 级扩展至 64 级。多片连接时，一个为主片，其余为从片，PC/AT 系统便采用这种级联工作方式。有关引脚说明如下：

$\overline{\text{SP}}/\overline{\text{EN}}$：从片编程/缓冲器允许信号线，是双功能引脚。作为输入信号使用时，用于区别 8259A 是主（$\overline{\text{SP}}=1$）还是从（$\overline{\text{SP}}=0$）。作为输出信号使用时，用于选通 8259A 至 CPU 之间的数据总线缓冲器。在只有一片 8259A 的系统中，$\overline{\text{SP}}/\overline{\text{EN}}$ 接高电平。

CAS_2～CAS_0：级联信号线。对于主片 8259A，它们是输出线，用于输出级联选择编码以选出请求中断的从片；对于从片 8259A，它们是输入线，接收主片送来的选择编码。

6.5.2　8259A 芯片的工作过程及工作方式

1．8259A 芯片的工作过程

当系统上电后，首先应由 CPU 对 8259A 芯片写入若干命令字对其进行初始化，使其处于准备就绪状态。当完成初始化后，8259A 就处于就绪状态，随时可接收外设送来的中断请求信号。当外设发出中断请求后，8259A 对外部中断请求的处理过程如下：

（1）当中断请求输入线 IR_7～IR_0 上有一条或若干条中断请求信号有效时，则 IRR 的相应位置 1。

（2）若中断请求线中至少有一条是中断允许的，则 8259A 由 INT 引脚向 CPU 发出中断请求信号。

（3）若 CPU 处于开中断状态，则在当前指令执行完成以后，用 $\overline{\text{INTA}}$ 信号作为响应。

（4）8259A 在接收到 CPU 发出的第一个 $\overline{\text{INTA}}$ 负脉冲后，使最高优先级的 ISR 位置 1，而相应的 IRR 位复位。在此中断响应周期中，8259A 并不向系统数据总线传送任何信息。

（5）CPU 在输出第二个 $\overline{\text{INTA}}$ 脉冲后，8259A 向数据总线输送一个 8 位的中断类型码。CPU 在此周期中，读取此类型码并乘以 4，就可以从中断向量表中取出中断服务程序的入口地址。据此，CPU 便可转入中断服务程序。

（6）若 8259A 工作在自动结束中断 AEOI（Automatic End Of Interrupt）模式，在第二

電子工業出版社

个 $\overline{\text{INTA}}$ 脉冲结束时，将使中断源在 ISR 中的相应位复位；否则，直到中断服务程序结束，由 CPU 向 8259A 发出 EOI 命令，才使 ISR 的相应位复位，表征对应此位的中断源全部服务完毕。

2．8259A 芯片的工作方式

8259A 芯片具有非常灵活的中断管理方式，这些方式都可以通过编程来设定。8259A 的工作方式有五种，具体见表 6-6。

（1）中断嵌套方式

按照优先权设置方法来分，8259A 有普通全嵌套和特殊全嵌套两种中断嵌套工作方式。

① 普通全嵌套方式。它是 8259A 最常用的工作方式，也称全嵌套方式。当芯片写入初始化命令字 ICW 后，如不再写入操作命令字 OCW，则自动进入并保持在这种方式。

② 特殊全嵌套方式。特殊全嵌套方式一般用在 8259A 级联系统中。这时，主片 8259A 编程为特殊全嵌套方式，其余编程为普通全嵌套方式。当来自某一从片的中断请求正在处理时，一方面，和普通全嵌套方式一样，对来自优先级较高的主片其他引脚上的中断请求进行开放；另一方面，对来自同一从片的较高优先级请求也会开放。这样，在同一从片的 ISR 中就会有不止一位置 1 的情况。

因此，多片级联系统中来自从片的中断服务结束时，须用软件检查刚刚结束的中断是否是从片的唯一中断，否则不能将主片 ISR 中的相应位复位。其方法是：先向从片发出一普通结束中断命令 EOI，然后读出 ISR 内容。若为 0，表示当前只有一个中断被服务，这时可再向主片发一个 EOI 命令；否则，说明该从片有两个或以上中断源，不应发给主片 EOI 命令。待该从片中断服务全部结束后，再发送 EOI 命令给主片。

（2）中断优先权循环方式

在实际应用中，中断源的优先权情况比较复杂，不一定有明显的等级，也不可能总是规定 IR_0 优先权最高，IR_7 优先权最低，应根据实际情况来具体处理。8259A 芯片设计了自动和特殊两种中断优先权循环方式。

① 自动循环方式。如初始优先级队列规定为 IR_0，IR_1，…，IR_7，此时若 IR_4 有中断请求则处理 IR_4。IR_4 得到服务后自动左循环到最低优先级，IR_5 成为最高优先级，中断源的优先级队列依次变为 IR_5，IR_6，IR_7，IR_0，IR_1，IR_2，IR_3，IR_4。

② 特殊循环方式。如指定 IR_5 为最低优先级，则当前的优先级顺序为 IR_6，IR_7，IR_0，IR_1，…，IR_5。

（3）中断结束处理方式

不管用哪种优先权方式工作，当一个中断请求得到响应时，8259A 都会将中断服务寄存器 ISR 中相应位置 1。当中断服务程序结束时，必须使该 ISR 位清 0。否则，这个未复位的高优先权中断源会影响 8259A 的正常控制功能。

使 ISR 位复位的动作就是对 8259A 进行中断结束处理。8259A 提供自动和非自动两种中断结束处理方式，其中非自动中断结束方式又包括普通（或称为一般、正常）中断结束方式和特殊中断结束方式。

再次强调，在多片级联系统非自动结束方式下，从片在中断服务程序结束时，必须发两次 EOI 命令，一次是对从片发送的，另一次则是对主片发送的。当工作于特殊全嵌套方式时，第二个向主片发送的 EOI 命令是否输出，取决于对从片的 ISR 检测是否为 0。

電子工業出版社

表 6-6　8259A 芯片的五种工作方式

功能分类		工作方式		工作方式描述
设置中断优先权	优先权固定	中断嵌套方式	普通全嵌套方式	中断优先级由高到低固定为：IR_0、IR_1、IR_2、IR_3、IR_4、IR_5、IR_6、IR_7 CPU 响应中断时，8259A 将申请中断的优先权最高的中断源在 ISR 中的相应位置 1，并服务该中断源。与它同级或低级的中断源申请被屏蔽，可响应优先级比它高的中断源的申请，适用于单片 8259 系统
			特殊全嵌套方式	中断优先级由高到低固定为：IR_0、IR_1、IR_2、IR_3、IR_4、IR_5、IR_6、IR_7 与普通嵌套方式基本相同，唯一区别是，当处理某级中断时，对同级的中断请求也会给予响应。适用于 8259 级联的情况
	优先权循环	中断优先权循环方式	自动循环方式	一个中断源得到服务后，它的优先级自动降为最低，原来比它低一级的中断源则为最高级，依次排列，适用于系统中多个中断源优先级相同的场合
			特殊循环方式	通过编程写 OCW2 的低 3 位来指定最低优先级，原来比它低一级的中断源则为最高级，依次排列，适用于中断源优先级需要任意改变的情况
中断结束处理		中断结束处理方式	自动中断结束方式（AEOI）	在 CPU 第二个中断响应周期 $\overline{\text{INTA}}$ 信号的后沿，8259A 自动将 ISR 中的相应位复位。适用于不要求中断嵌套的场合
			普通中断结束方式（EOI）	当 8259A 工作在该方式下，当前服务过的中断源就是中断优先权最高的源，用普通的 EOI 命令使它在 ISR 中的相应位复位。这个命令应加在中断服务程序的末尾处，适用于普通全嵌套工作方式。
			特殊中断结束方式（SEOI）	当 8259A 工作在非普通全嵌套工作方式时，由于中断优先级不断改变，8259A 无法确定刚服务是哪级中断，就要由 CPU 输出 SEOI 命令，指出要清除哪个 ISR 位
屏蔽中断源		屏蔽中断源方式	普通屏蔽方式	编程写 OCW1 将 IMR 某位置 1，则屏蔽对应的中断源。该位清 0，则允许该级中断
			特殊屏蔽方式	用于开放较低级中断请求
中断触发		中断触发方式	电平触发方法	8259A 的引脚 IR_n（n=0～7）上出现高电平，表示有中断请求。此方式下，应注意响应后及时去除高电平，否则可能引起不应该有的第二次中断
			边沿触发方式	8259A 的引脚 IR_n（n=0～7）上出现上升沿跳变，表示有中断请求

（4）屏蔽中断源方式

8259A 的 8 个中断请求都可根据需要单独屏蔽，屏蔽就是通过编程使屏蔽寄存器 IMR 中相应位清 0 或置 1，以允许或禁止相应的中断请求。8259A 有普通和特殊两种屏蔽方式。

有时希望在执行一个较高级的中断服务过程中，开放对较低级中断源的服务。为此，自然会想到使 IMR 的相应位置 1，使本级中断受到屏蔽，为开放较低级中断请求提供可能。但是，在 8259A 这样做有一个问题。因为每当一个中断请求得到响应时，就会使 ISR 相应位置 1，只要没有收到 EOI 命令，8259A 就会据此而禁止所有优先级比它低的中断。所以，尽管当前处理的较高级中断请求被屏蔽，但由于 ISR 位未被复位，较低级的中断请求在发出 EOI 命令之前仍不会得到响应。

8259A 的特殊屏蔽方式可以解决这个问题。首先屏蔽当前中断级（IMR 对应位为 1），然后设置 8259A 为特殊屏蔽方式，则使 ISR 相应位的功能中止，直到为较低级中断服务完后，再清除特殊屏蔽方式为止。通过设置特殊屏蔽方式，可动态改变中断系统的优先级结构。

（5）中断触发方式。8259A 的中断触发方式有电平触发和边沿触发两种方式。无论是电平触发还是边沿触发，中断请求信号 IR_n 都应维持足够的宽度。即在第一个中断响应信号 $\overline{INTA}$ 结束之前，IR_n 都必须保持高电平。如果 IR_n 信号提前变为低电平，8259A 就会默认为这个中断请求来自引脚 IR_7，这种办法能够有效地防止由 IR_n 输入端严重的噪声尖峰而产生的中断。因此，对应 IR_7 的中断服务程序可只执行一条返回指令，从而滤除这种中断。如果 IR_7 另有他用，仍可通过读 ISR 状态来识别非正常的 IR_7 中断，正常的 IR_7 中断会使相应的 ISR 的 D_7 位置 1。

6.5.3 8259A 命令字

8259A 是可编程中断控制器，对其应用编程包括初始化编程和操作方式编程两部分。

① 初始化编程。由 CPU 向 8259A 送 2～4 字节的初始化命令字 ICW（Initialization Command Word）。在 8259A 工作之前，必须写入初始化命令字使其处于准备就绪状态，写入后一般不再改变。

② 操作方式编程。由 CPU 向 8259A 送 3 字节的操作命令字 OCW（Operation Command Word），以规定 8259A 的操作方式。OCW 可在 8259A 已经初始化以后的任何时间内写入。

下面分别介绍 8259A 的初始化命令字及操作命令字的格式与功能。

1．8259A 的初始化命令字 ICW

初始化命令字 ICW 最多有 4 个，必须在 8259A 开始工作之前写入。

写入 ICW1 的端口地址规定为 A_0=0（偶地址）。写入 ICW2、ICW3 和 ICW4 的端口地址均为 A_0=1（奇地址），由于片内采用了 FIFO 缓冲器技术，编程时需严格按照图 6-28 所示的初始化流程顺序写入。

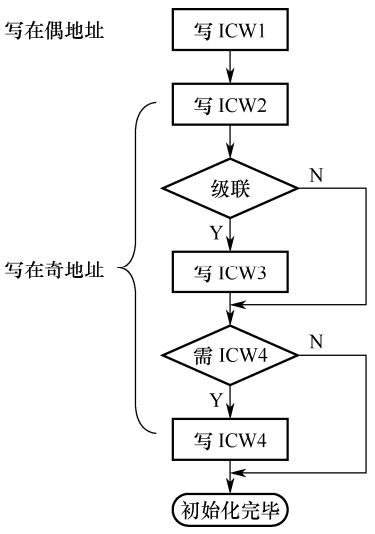


图 6-28　初始化流程

電子工業出版社

（1）ICW1（初始化字）

8259A 最初写入的必须是 ICW1。当引脚 A_0=0 和 ICW1 内的 D_4=1 时，表示写入的是 ICW1 命令字。ICW1 写入后，除完成 ICW1 格式规定的功能，8259A 内部状态有如下初始化过程：

① 对中断请求信号边沿检测电路复位，准备按 ICW2、ICW3、ICW4 顺序接收其余的 ICW。

② 清除 ISR 和 IMR。

③ 指定优先级由高到低为：IR_0，IR_1，…，IR_7。

④ 特殊屏蔽方式复位，即设定为普通屏蔽方式。

⑤ 自动中断结束方式复位，即采用非自动中断结束方式。

⑥ 状态读出电路预置为 IRR。

ICW1 的格式如图 6-29 所示。

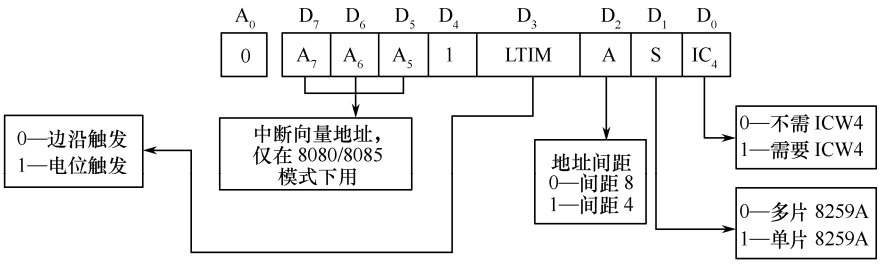


图 6-29　ICW1 的格式

D_0（IC4）：表示初始化过程要不要写 ICW4，在 80x86 系统中 D_0=1。

D_1（S）：指明系统中是使用单片还是多片 8259A，如果是多片级联需要写入 ICW3。

D_3（LTIM）：说明中断请求信号起作用的触发方式。

D_2 和 D_7～D_5：只在 8080/8085 CPU 模式下使用，指明 8 个中断向量地址之间的间距和中断向量地址。8086/8088 模式下不用，通常将它们置为 0。

（2）ICW2（中断类型码字）

ICW2 是中断类型码设置寄存器，写入时应 A_0=1，ICW2 的格式如图 6-30 所示。

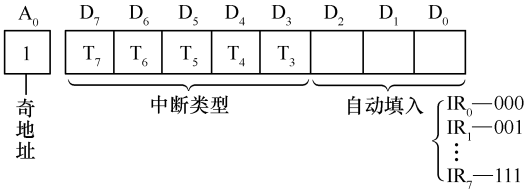


图 6-30　ICW2 的格式

工作在 8080/8085 系统中时，8 位全部有用。在与 8086/8088 CPU 连接使用时，用 T_7～T_3 作为中断类型码的高 5 位，低 3 位由 8259A 自动按 IR_n 输入端确定，如表 6-7 所示。

如在 PC/XT 和 AT 中，T_7～T_3=00001，T_2～T_0=000，ICW2=08H，表明 IR_0～IR_7 的中断类型码分别为 08H～0FH。在 8259A 收到第二个 $\overline{INTA}$ 时，将中断类型码送到数据总线上。

（3）ICW3（级联控制字）

ICW3 定义了 8259A 的级联。若系统中只有一片 8259A，则不用 ICW3；若有多片 8259A 级联，则主 8259A 和每一片从 8259A 都必须使用 ICW3。主、从片 8259A 的 ICW3 是不同的，写入 ICW3 时应 A_0=1，ICW3 的格式如图 6-31 所示。

電子工業出版社

表 6-7　8259A 安排中断类型码

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
IR_7	T_7	T_6	T_5	T_4	T_3	1	1	1
IR_6	T_7	T_6	T_5	T_4	T_3	1	1	0
IR_5	T_7	T_6	T_5	T_4	T_3	1	0	1
IR_4	T_7	T_6	T_5	T_4	T_3	1	0	0
IR_3	T_7	T_6	T_5	T_4	T_3	0	1	1
IR_2	T_7	T_6	T_5	T_4	T_3	0	1	0
IR_1	T_7	T_6	T_5	T_4	T_3	0	0	1
IR_0	T_7	T_6	T_5	T_4	T_3	0	0	0

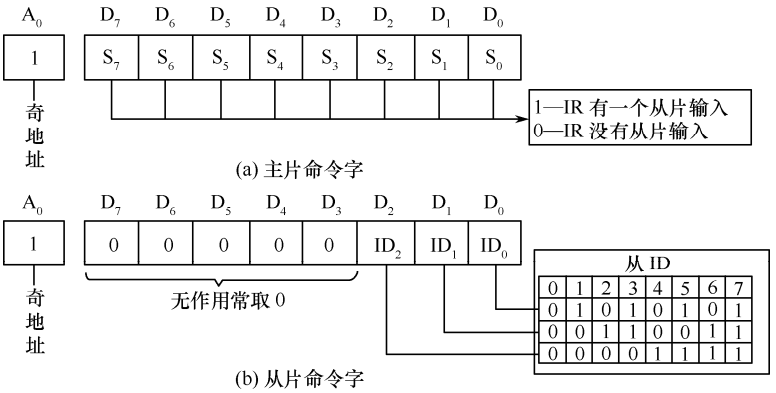


图 6-31　ICW3 的格式

在主片 ICW3 中的 D_7～D_0 位表示相应的 IR_7～IR_0 中断请求线上有无从片，某位 D_i= 1 表示对应的 IR_i 线上有从片。从片的 ICW3 用 D_2～D_0（ID_2～ID_0）位来表示从片的识别码，它对应于主片 IR_7～IR_0 级联的从片编码。D_7～D_3 没有定义，一般取 0。

（4）ICW4（中断结束方式字）

ICW4 定义 8259A 工作时所用的 CPU 类型，以及中断服务程序是否送出 EOI 命令以清除 ISR 中的相应位。写入 ICW4 时应 A_0=1，ICW4 格式如图 6-32 所示。

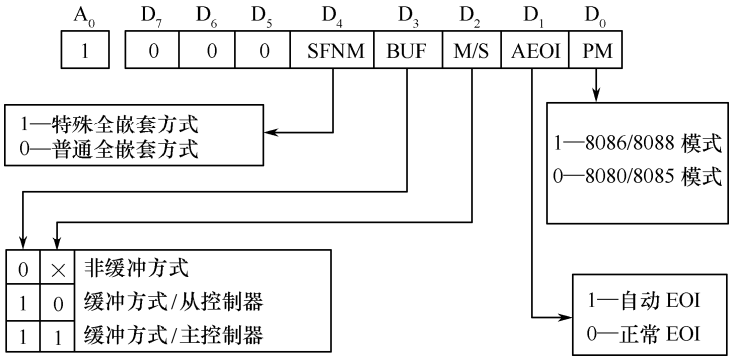


图 6-32　ICW4 的格式

D_0（PM）：定义选用的处理器类型。

D_1（AEOI）：表示是否有自动结束中断、使 ISR 复位的功能。若是 0，表示没有，需要在中断服务程序的末尾送出 EOI 命令使 ISR 相应位复位；若是 1，表示 8259A 能自动使 ISR

電子工業出版社

复位。

D_2（M/S）：表示本片 8259A 是主片（M/S=1）还是从片（M/S=0）。

D_3（BUF）：表示本片 8259A 和系统数据总线间是否有缓冲器，1 表示有。因此必须产生控制信号，以便中断时能打开缓冲器。

D_4（SFNM）：表示 8259A 是否为多片级联系统中的主片，1 表示是，其优先级顺序采用特殊全嵌套方式；0 表示否，其优先级顺序采用普通全嵌套方式。

D_7～D_5：未用，一般取 0。

在写完初始化命令字 ICW 后，8259A 的中断输入端就可以接收中断请求信号了。若不再写入任何操作命令字 OCW，8259A 便处于全嵌套中断工作方式。这时，中断源的优先级固定为 IR_0 最高，IR_7 最低。当 CPU 为高级中断服务时，则将 ISR 对应的位置 1，这时，8259A 不再响应所有同级或低级的中断请求，直到处理完高级中断，再执行一条 EOI 命令为止。如果 CPU 正在为低级中断源服务，则此时 ISR 相应位置 1，当有较高级别中断请求时，如果 CPU 处于开中断状态（IF=1），便允许响应此高级中断，产生中断嵌套。此时 8259A 便将对应此高级中断的 ISR 相应位置 1，原来较低级中断的 ISR 相应位并不复位，只是将该低级中断暂时挂起，转向为高级中断服务。当为高级中断服务完毕，程序中发一条 EOI 命令和中断返回命令 IRET 时，高级中断源对应的 ISR 相应位才复位，程序返回到低级中断服务程序的断点处，若没有更高级的中断申请时，则被挂起的低级中断又从断点处开始执行。

若需改变上述 8259A 的中断控制方式，或为了屏蔽某些中断，以及读出 8259A 的一些状态信息，如 IRR、ISR、IMR 的内容，则必须写入操作命令字 OCW。

2．8259A 的操作命令字 OCW

8259A 的操作命令字有 3 个：OCW1～OCW3。在编程写入操作命令字时，顺序上没有严格的要求，需要什么字就写入什么字。但是，严格规定 OCW1 写入奇地址端口（A_0=1）；OCW2 和 OCW3 都写入偶地址端口（A_0=0），由字中的特征位 D_4D_3 来区别它们是不同的命令字。

（1）OCW1（屏蔽控制字）

设置或清除对中断源的屏蔽，亦称中断屏蔽命令字。当写入 OCW1 使 IMR 的某位为 1 时，则相应位的中断源被屏蔽；若为 0，则中断被允许。如 OCW1=80H，则表示 IR_7 中断源被屏蔽，即使 IR_7 引脚有中断请求，但由于中断屏蔽寄存器 IMR 的 D_7 被置 1，故 8259A 并不发出 INT 信号。

OCW1 的格式如图 6-33 所示。利用 OCW1，可以在程序的任何地方实现对某些中断的屏蔽或开放。

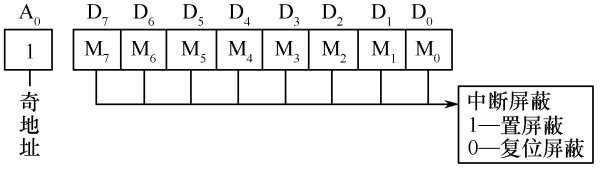


图 6-33　OCW1 的格式

（2）OCW2（中断结束和优先级循环控制字）

写入 OCW2 主要是对 8259A 发出中断结束命令，包括普通结束 EOI 和特殊结束 SEOI，并控制中断优先权的循环等。OCW2 的特征位 D_4D_3=00，其格式如图 6-34 所示。

電子工業出版社

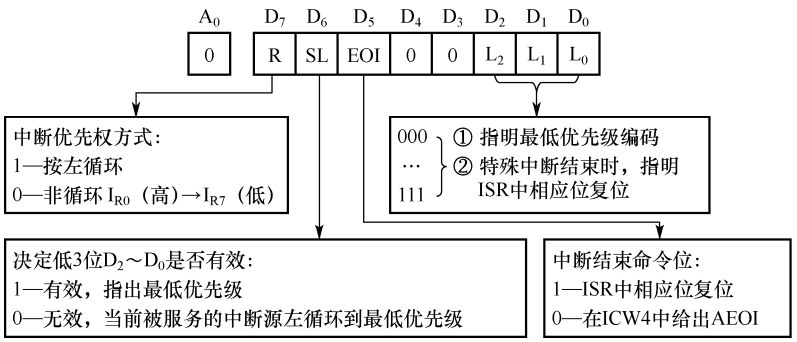


图 6-34　OCW2 的格式

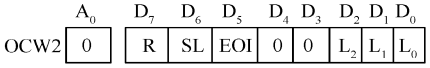
EOI：中断结束命令，该位为 1 时，使 ISR 中当前最高优先级的相应位复位，以便允许系统再为其他级别中断源服务。如果 ICW4 的 AEOI=0，则必须在中断服务程序的返回指令 IRET 前，写一条 OCW2 命令字，以给出 EOI 标志。8259A 得到 EOI 命令后，便自动将为此中断服务的 ISR 中的相应位复位。

R：中断优先权是否循环的标志。R=0 时，8 个中断请求的优先级固定不变，如 IR_7 最低，IR_0 最高。R=1 时，优先级可以循环。优先级循环是指中断源优先级采用左循环轮转方式，当一个优先级别的中断服务完毕后，如 IR_4，它就左循环到最低的优先级，而和它相邻的 IR_5 变为最高优先级，IR_6 变成次最高级，其他以此类推。

$L_2L_1L_0$：这三位的第一个作用是设定系统中最低优先级的编码，可用来改变 8259A 复位时所设置的 IR_7 最低、IR_0 最高的优先级规定。第二个作用是在 OCW2 给出特殊中断结束命令 SEOI 时，指明具体要特殊结束哪个中断级，使 ISR 中的相应位复位。

SL：是选择 $L_2L_1L_0$ 编码是否有效的标志。SL=1，则 $L_2L_1L_0$ 选择有效。此时若 R=1，则优先权左循环至最低优先级对准 $L_2L_1L_0$ 编码 i（$0 \leqslant i \leqslant 7$）表示的 IR_i 这一特殊值为止。SL=0，$L_2L_1L_0$ 选择无效。此时若 R=1，当前服务的中断请求自动循环到最低优先级。

OCW2 中 R、SL、EOI 的不同组合表征了 8259A 的不同工作方式，凡 $L_2L_1L_0$ 选择有效都称为“特殊”，可对照图 6-35 进一步深入理解。



R SL EOI	功　　能
0 0 1	普通EOI命令，全嵌套方式
0 1 1	特殊EOI命令，全嵌套方式，L_2~L_0指定对应ISR位清0
1 0 1	普通EOI命令，优先级自动循环
1 1 1	普通EOI命令，优先级特殊循环，L_2~L_0指定优先级最低的IR
1 0 0	自动EOI时，优先级自动循环
0 0 0	自动EOI时，取消优先级自动循环
1 1 0	优先级特殊循环，L_2~L_0指定优先级最低的IR
0 1 0	无操作

图 6-35　OCW2 的格式

（3）OCW3（屏蔽和读状态控制字）

OCW3 可设置特殊屏蔽方式、查询方式，以及读取 8259A 中 IRR、ISR、IMR 的当前状态。OCW3 的特征位 D_4D_3=01，格式如图 6-36 所示。

電子工業出版社

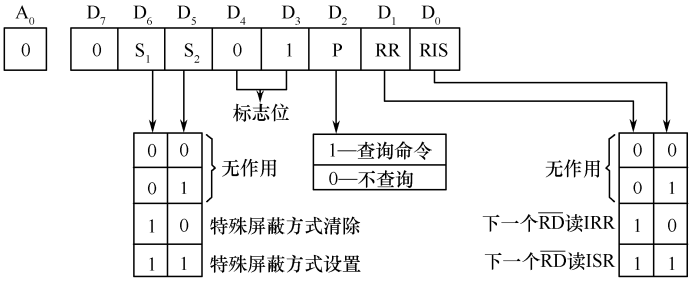


图 6-36　OCW3 的格式

① 设置特殊屏蔽方式。D_6D_5（ESMM 和 SMM）两位用于设置特殊屏蔽方式。如果在执行高级中断服务程序时需要开放低级中断，可先通过 OCW1 将正在被服务的高级中断屏蔽，再用 D_6D_5=11 的 OCW3 命令设置 8259A，则可使 ISR 相应位的功能中止，直到清除特殊屏蔽方式（D_6D_5=10）为止。

② 查询中断请求。在 CPU 内部禁止中断时，或者在不想用 INT 引脚向 CPU 申请中断时，可以使用 8259A 的查询工作方式。

写入 P=1 的 OCW3 就发出查询命令。8259A 得到查询命令后，立即组成查询字，等待 CPU 来读取。所以，当 CPU 再对同一地址（A_0=0）执行一条输入指令时，便可读到如下查询字：

$$I \quad \times \quad \times \quad \times \quad \times \quad W_2 \quad W_1 \quad W_0$$

若 I = 1，则表示本片 8259A 的 IR_0～IR_7 中有中断请求，对应当前优先级最高的外部中断请求的编码由 $W_2W_1W_0$ 给出。若 I = 0，则表示无中断请求。

③ 读取 8259A 的状态。为了解 8259A 的工作状态，CPU 可读出 IRR、ISR、IMR 三个寄存器的内容，但读的方法有所不同。

读 IRR、ISR。先写入读命令字 OCW3 到偶地址（A_0=0）端口，再用 IN 指令从偶地址（A_0=0）端口读出 IRR/ISR。读命令字 OCW3 的格式为 0　0　0　0　1　0　R_1　R_0。

读 IMR。不需写入 OCW3 命令，只要直接对奇地址（A_0 = 1）端口进行读操作即可。

综上所述，写入 8259A 的命令字共有 7 个，从 8259A 读出的状态字有 4 个，它们是由读 $\overline{RD}$ 和写 $\overline{WR}$ 信号、地址信号 A_0 以及命令字中的某些特定位所区分的，如表 6-8 所示。

表 6-8　8259A 的命令字/状态字读/写条件

A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	XT 机地址	AT 机扩展从片地址	功　能
0	1	0	0	20H	0A0H	写入 ICW1、OCW2、OCW3（注①）
1	1	0	0	21H	0A1H	写入 ICW2、ICW3、ICW4、OCW1（注②）
0	0	1	0	20H	0A0H	读出 IRR、ISR 和查询字（注③）
1	0	1	0	21H	0A1H	读出 IMR
×	1	1	0			数据总线高阻状态
×	×	×	1			数据总线高阻状态

注：① 由控制字中的 D_4D_3 标志位决定写入的是 OCW2 还是 OCW3。② 写入 ICW1 后，由片内的顺序逻辑确定后续 ICW，否则写入 OCW1。③ 由 OCW3 的内容决定读出哪个状态字。

6.5.4　8259A 芯片应用举例

1．8259A 在 PC/XT 机中的应用

在 IBM PC/XT 机中，只使用一片 8259A 管理可屏蔽中断，如图 6-37 所示。

電子工業出版社

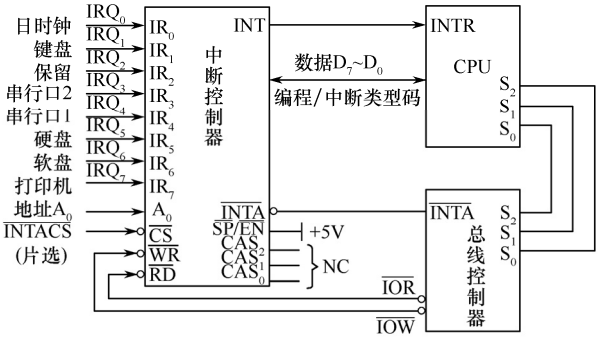


图 6-37　PC/XT 8259A 硬件连接

IRQ_0 接至系统板上定时/计数器 Intel 8253 通道 0 的输出信号 OUT_0，用作微机系统的日时钟中断请求；IRQ_1 是键盘输入接口电路送来的中断请求信号，用来请求 CPU 读取键盘扫描码；IRQ_2 是系统保留的；另外 5 个请求信号 IRQ_3～IRQ_7 来自 I/O 通道扩展板电路，分别是串行通信接口 2、串行通信接口 1、硬盘与软盘适配器、并行打印机的中断请求信号。

在 PC/XT 的 I/O 地址空间中，分配给 8259A 的 I/O 端口地址范围为 20 H～3FH，通常偶、奇端口地址取 20H、21H。PC/XT 对 8259A 的初始化设置为：边沿触发中断、缓冲器方式、普通 EOI 命令结束中断、中断优先权管理采用全嵌套方式，8 级中断源的类型码为 08H～0FH。各级中断源规定如表 6-9 所示。

表 6-9　IBM PC/XT 中 8259A 的 8 级中断分配表

中断向量地址指针	引脚	中断类型码	中断源	中断向量地址指针	引脚	中断类型码	中断源
0020H	IRQ_0	08H	定时器	0030H	IRQ_4	0CH	串行口 1
0024H	IRQ_1	09H	键盘	0034H	IRQ_5	0DH	硬盘
0028H	IRQ_2	0AH	用户保留	0038H	IRQ_6	0EH	软盘
002CH	IRQ_3	0BH	串行口 2	003CH	IRQ_7	0FH	并行打印机

（1）8259A 初始化编程

根据系统要求，对 8259A 进行如下初始化编程。完成后，8259A 为全嵌套工作方式，可以响应外部中断请求。

```
MOV     AL, 00010011B      ; 设置 ICW1 为边沿触发，单片 8259A，需要 ICW4
OUT     20H, AL
MOV     AL, 00001000B      ; 设置 ICW2 中断类型码基数为 08H，则可响应的
OUT     21H, AL            ; 8 个中断类型码为 08H～0FH。
MOV     AL, 00001101B      ; 设置 ICW4 为 8086/8088 模式，普通 EOI，缓冲
OUT     21H, AL            ; 方式，全嵌套方式。
```

（2）8259A 操作方式编程

在用户程序中，可以用 OCW1 来设置中断屏蔽寄存器 IMR，以允许或屏蔽各外设的中断申请，但注意不要破坏系统原设定工作方式。如允许日时钟 IRQ_0 和键盘 IRQ_1 中断，其他中断状态不变，则可送入以下指令：

```
IN      AL, 21H            ; 读出 IMR
AND     AL, 0FCH           ; 只允许 IRQ0 和 IRQ1，其他不变
OUT     21H, AL            ; 写入 OCW1，即设定 IMR
```

由于中断结束采用非自动结束方式，因此在中断服务程序结束返回断点前，必须对

電子工業出版社

OCW2 写入 00100000B（20H），发出中断结束命令。

```
MOV     AL, 20H                 ; 设置 OCW2 的值为 20H
OUT     20H, AL                 ; 写入 OCW2 的端口地址 20H
…                               ; 恢复现场
IRET                            ; 中断返回
```

在程序中，通过设置 OCW3 可读出 IRR、ISR 的状态。如要读出 IRR 内容以查看申请中断的中断源：

```
MOV     AL, 0AH                 ; 先写入 OCW3，读 IRR 命令
OUT     20H, AL
NOP                             ; 延时，等待 8259A 操作结束
IN      AL, 20H                 ; 读出 IRR
```

IMR 的内容可以随时方便地从 A_0=1 端口读出。如在 BIOS 中，对 IMR 的检查程序如下：

```
MOV     AL, 0                   ; 设置 OCW1 为 0，送 OCW1 端口地址，表示 IMR 为全 0
OUT     21H, AL
IN      AL, 21H                 ; 读 IMR 状态
OR      AL, AL
JNZ     ERR                     ; 若不为 0，则转出错程序 ERR
        MOV     AL, 0FFH        ; 设置 OCW1 为 FFH，送 OCW1 端口地址，表示 IMR 为全 1
        OUT     21H, AL
        IN      AL, 21H         ; 读 IMR 状态
        ADD     AL, 1           ; IMR=0FFH?
        JNZ     ERR             ; 若不是 0FFH，则转出错程序 ERR
        …
ERR:    …
```

2．8259A 在 PC/AT 机中的应用

在 PC/AT 机中，由两片 8259A 以主、从级联方式管理 15 级中断，如图 6-38 所示。

级联时，主、从片的 CAS_2～CAS_0 互连，从片的 INT 端输出连至主片的 IRQ_2 输入。主片的端口地址为 20H、21H，中断类型码为 08H～0FH；从片的端口地址为 A0H、A1H，中断类型码为 70H～77H。其中，IRQ_0 仍用于日时钟中断（08H），IRQ_1 仍用于键盘中断（09H）。扩展的 IRQ_8 用于实时时钟中断，IRQ_{13} 来自协处理器 80287。除此之外，所有其他中断请求信号都来自 I/O 通道的扩展板。

（1）8259A 初始化编程。

对主片 8259A 的初始化：

```
MOV     AL, 11H         ; 写入 ICW1，设定边沿触发，级联方式
OUT     20H, AL
NOP                     ; 延时，等待 8259A 操作结束，下同
MOV     AL, 08H         ; 写入 ICW2，设定 IRQ0 的中断类型码为 08H
OUT     21H, AL
NOP
MOV     AL, 04H         ; 写入 ICW3，设定主片 IRQ2 级联从片
OUT     21H, AL
NOP
MOV     AL, 11H         ; 写入 ICW4，设定特殊全嵌套方式，普通 EOI 方式
```


電子工業出版社

```
OUT     21H, AL
```

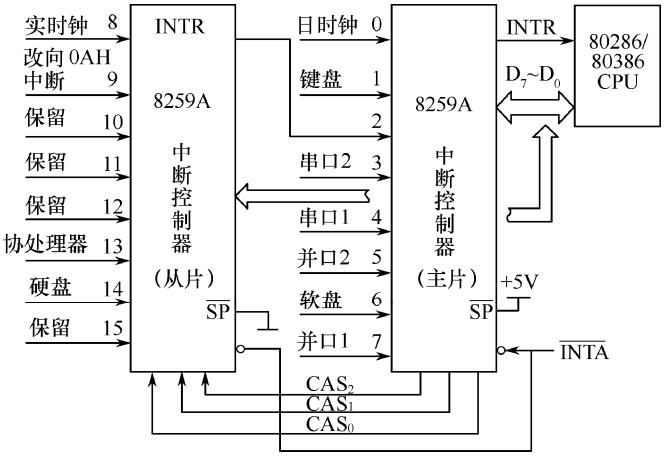


图 6-38　PC/AT 8259A 硬件连接

对从片 8259A 的初始化：

```
MOV     AL, 11H         ; 写入 ICW1，设定边沿触发，级联方式
OUT     0A0H, AL
NOP
MOV     AL, 70H         ; 写入 ICW2，设定从片 IR0，即 IRQ8 的中断类型码为 70H
OUT     0A1H, AL
NOP
MOV     AL, 02H         ; 写入 ICW3，设定从片级联于主片的 IRQ2
OUT     0A1H, AL
NOP
MOV     AL, 01H         ; 写入 ICW4，设定普通全嵌套方式，普通 EOI 方式
OUT     0A1H, AL
```

（2）级联工作编程

级联时，主片工作在特殊全嵌套方式。当来自某个从片的中断请求进入服务时，主片的优先权控制逻辑不封锁这个从片，从而使来自该从片的更高优先级的中断请求能被主片所识别，并向 CPU 发出中断请求。因此中断服务程序结束时必须用软件来检查被服务的中断是否是该从片中唯一的中断请求，流程如下：

先向从片发出一个 EOI 命令，清除已完成服务的 ISR 位。再读出 ISR 的内容检查它是否为 0。如果 ISR 的内容为 0，则向主片发一个 EOI 命令清除与从片相对应的 ISR 位；否则，不向主片发 EOI 命令，继续进行从片的中断处理，直到 ISR 的内容为 0，再向主片发出 EOI 命令。

读 ISR 的内容：

```
MOV     AL, 0BH             ; 写入 OCW3，读 ISR 命令
OUT     0A0H, AL
NOP                         ; 延时，等待 8259A 操作结束
IN      AL, 0A0H            ; 读出 ISR
```

向从片发 EOI 命令：

```
MOV     AL, 20H
OUT     0A0H, AL            ; 写从片 EOI 命令
```


電子工業出版社

向主片发 EOI 命令：

```
MOV     AL, 20H
OUT     20H, AL              ; 写主片 EOI 命令
```

6.6 中断程序设计

6.6.1 设计方法

80x86 系列微机可以处理 256 个中断，分为内部与外部中断两种类型。对不同类型中断的处理过程有些差异，但中断服务程序的编程原则大致相同。由于可屏蔽中断涉及对中断控制器的操作，所以其编程情况较内部中断和非屏蔽中断复杂些。在此以可屏蔽中断为例，介绍中断程序设计的一般过程（假设采用的中断控制器是 8259A）。

1. 设置中断向量表

当外设申请中断得到响应后，CPU 首先在中断向量表中查找中断服务程序的入口地址，以便转入中断服务程序。所以在主程序中需设置中断向量，使其指向相应的中断服务程序。

如果利用 PC 已有的内部 8259A 处理中断，必须在设置中断向量表之前，首先保存系统原中断向量的内容，待用户程序执行完毕退出之前，取出原中断向量写入到中断向量表中，以恢复系统原始状态。

设置中断向量表可以利用传送指令直接访问中断向量表的相应存储单元，写入中断向量；PC 也可利用 DOS 系统功能调用 INT 21H 的 25H 和 35H 子功能修改中断向量。假如借用 PC/XT 中的 IRQ_3（中断码 0BH）响应外部中断，中断后须执行的中断服务程序名为 INTPROC，其中断向量表的设置可以用下面的程序段实现：

```
; 保存原中断向量的内容
INTSEG   DW   ?              ; 定义存放原中断向量的单元
INTOFF   DW   ?
…
MOV     AH, 35H              ; 读取原中断向量在 ES:BX 中
MOV     AL, 0BH
INT     21H
MOV     INTSEG, ES           ; 保存原中断向量的段基址
MOV     INTOFF, BX           ; 保存原中断向量的偏移量
; 重新修改中断向量的内容
CLI                          ; 关中断，设置中断向量新内容
PUSHDS
MOV     AX, SEG  INTPROC
MOV     DS, AX
MOV     DX, OFFSET  INTPROC
MOV     AH, 25H              ; 将 DS:DX 置于中断向量表中
MOV     AL, 0BH
INT     21H
POP     DS
```


電子工業出版社

2．设置中断控制器

主程序中还需对中断控制器 8259A 进行初始化设置。若利用 PC 内部的 8259A 处理中断，由于操作系统已对 8259A 初始化，所以只需对 IMR 进行相应处理，开放那些允许的中断请求即可。同样，为了使应用程序返回 DOS 后系统能恢复原状态，应在修改 IMR 之前保存其原有内容，并于程序退出前予以恢复。假设允许 IRQ_3 响应外部中断，编程修改 IMR 的程序段如下：

```
INTIMR   DB   ?                ; 定义存放原 IMR 的单元
…
IN       AL, 21H              ; 读出 IMR
MOV      INTIMR, AL           ; 保存原 IMR 内容
AND      AL, 0F7H             ; 允许 IRQ3，其他不变
OUT      21H, AL              ; 设置新 IMR 内容
…
; 恢复原先的 IMR
MOV      AL, INTIMR           ; 取出保留的 IMR 原内容
OUT      21H, AL              ; 重写 OCW1
```

3．设置 CPU 的中断允许标志 IF

硬件中断是随机的，当外设提出可屏蔽中断请求，系统是否允许响应这个中断可以通过设置 CPU 中的中断允许标志位 IF 来控制，设置 IF=0 或 IF=1 的关中断和开中断指令分别是 CLI 和 STI。通常，在系统刚上电需要初始化时不能产生中断，这段时间必须关中断；初始化完毕则应开中断，以便及时响应中断请求为外设提供服务。另外，进入中断服务程序后，也可马上开中断，以允许较高级的中断能够嵌套。

4．设计中断服务程序

无论中断服务程序完成什么服务，其中须含有：保护现场、中断服务、向 8259A 发送中断结束命令、恢复现场、中断返回的程序段。程序段的一般结构如下：

```
INTPROC     PROC                ; 中断服务程序
            PUSH    AX          ; 保护现场
            PUSH    BX
            …
            STI                 ; 开中断，以允许中断嵌套
            …                   }
            …                   } ; 中断处理
            CLI                 ; 关中断
            MOV     AL, 20H     ; 向 8259A 发送 EOI 命令
            OUT     20H, AL
            POP     BX          ; 恢复现场
            POP     AX
            IRET                ; 中断返回
INTPROC     ENDP
```

外部中断服务程序是处理较紧急的事务的，因此服务时间应尽量短，能放在主程序完成的任务就不要由中断服务程序完成。另外，外部中断服务程序不要使用 DOS 系统功能调用 INT 21H，因为 DOS 版本的内核是不可重入的。例如，当主程序执行一个 DOS 系统功能调用时产生了外部中断，中断服务程序又调用这个 DOS 系统功能，就出现了 DOS 重入，

電子工業出版社

这是不允许的。在中断服务程序中若要控制 I/O 设备，最好调用 ROM BIOS 功能子程序或对 I/O 端口直接编程。

上述中断程序设计过程，除了对中断控制器 8259A 的操作外，其他均适合内部中断和非屏蔽中断。但非屏蔽中断在 PC 中有特殊作用，一般不要改写其中断服务程序。

6.6.2 中断程序设计举例

1．软中断程序设计举例

【例 6-5】 自己定义一个软中断，中断类型码为 80H，在中断服务程序中完成 ASCII 码加偶校验位（第 7 位）的工作。ASCII 码首地址为 ASCBUF，字节数为 COUNT，加偶校验位后仍放回原处。程序如下：

```
DATA    SEGMENT                          ; 数据段
        ASCBUF  DB  'ABCDEFGHIJ1234567890'
        COUNT   EQU  $-ASCBUF
DATA    ENDS
STACK   SEGMENT STACK   'STACK'          ; 堆栈段
        DB      100 DUP (?)
STACK   ENDS
CODE    SEGMENT                          ; 代码段
        ASSUME  CS:CODE, SS:STACK, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        CLI                              ; 关中断，设置中断向量表
        SUB     AX, AX
        MOV     ES, AX                   ; 中断向量表的段基址为 0000H
        MOV     DI, 4×80H                ; 在中断向量表中的偏移量
        MOV     AX, OFFSET  INTSV
        CLD
        STOSW                            ; 写中断向量的偏移量
        MOV     AX, SEG  INTSV
        STOSW                            ; 写中断向量的段基址
        STI
        …
        INT     80H                      ; 软中断指令
        …
        MOV     AH, 4CH                  ; 返回 DOS
        INT     21H
INTSV   PROC                             ; 中断服务程序
        PUSH    AX                       ; 保护现场
        PUSH    BX
        PUSH    CX
        MOV     CX, COUNT
        MOV     BX, OFFSET  ASCBUF
L2:     MOV     AL, [BX]
        AND     AL, AL                   ; 建立标志 PF
        JP      L1                       ; PF=1，偶数个 1
        OR      AL, 80H                  ; PF=0，奇数个 1
        MOV     [BX], AL                 ; 加入偶校验后写回
```


電子工業出版社

```
L1:     INC     BX
        LOOP    L2
        POP     CX                          ; 恢复现场
        POP     BX
        POP     AX
        IRET                                ; 中断返回
INTSV   ENDP
CODE    ENDS
        END     START
```

2．硬中断程序设计举例

【例 6-6】8259A 的 IRQ_0 中断请求（对应中断类型码 08H）来自定时器 8253 的通道 0。系统中设置它每隔 55 ms 产生一次中断，并提供完成日时钟计时功能的中断服务程序。本例中将替换系统计时程序，使得每次中断显示一串信息，显示 10 次后中止，程序返回 DOS。

```
STACK   SEGMENT  STACK              ; 堆栈段
        DB  256  DUP (0)
STACK   ENDS
DATA    SEGMENT                     ; 数据段
MSG     DB       'A 8259A INTERRUPT!', 0DH, 0AH    ; 显示信息，共 20 个字符
COUNTER DB       0                  ; 中断次数记录单元
INTSEG  DW       ?                  ; 保存原中断服务程序的段基址
INTOFF  DW       ?                  ; 保存原中断服务程序的偏移量
INTIMR  DB       ?                  ; 保存原 IMR 内容
DATA    ENDS
CODE    SEGMENT                     ; 代码段
        ASSUME  CS:CODE, DS:DATA, SS:STACK
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AX, 3508H           ; 保存原中断向量内容
        INT     21H
        MOV     INTSEG, ES
        MOV     INTOFF, BX
        CLI                         ; 关中断
        PUSH    DS                  ; 设置中断向量表新内容
        MOV     AX, SEG  INTPROC
        MOV     DS, AX
        MOV     DX, OFFSET  INTPROC
        MOV     AX, 2508H
        INT     21H
        POP     DS
        IN      AL, 21H             ; 读出 IMR
        MOV     INTIMR, AL          ; 保存原 IMR 内容
        AND     AL, 0FEH            ; 允许 IRQ0，其他不变
        OUT     21H, AL             ; 设置新 IMR 内容
        MOV     COUNTER, 0          ; 设置中断次数初值
        STI                         ; 开中断
L1:     CMP     COUNTER, 10         ; 循环等待中断
        JB      L1                  ; 中断 10 次退出
        CLI                         ; 关中断
        MOV     AL, INTIMR          ; 恢复 IMR
```


電子工業出版社

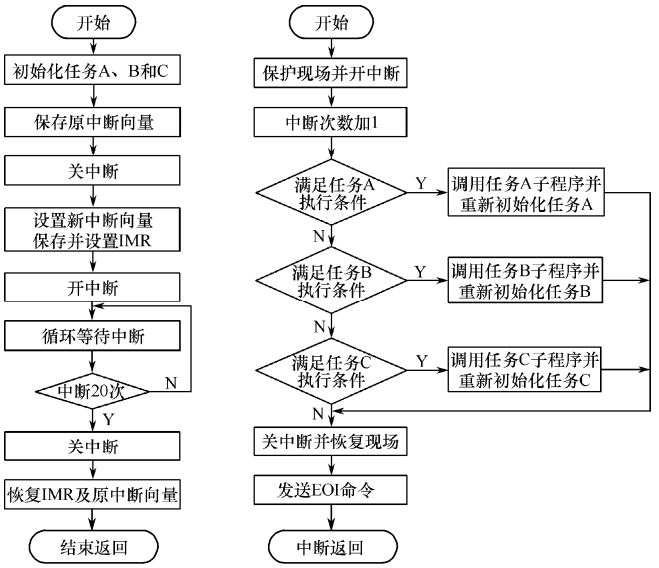
```
            OUT     21H, AL
            MOV     DX, INTOFF          ; 恢复中断向量内容
            MOV     AX, INTSEG
            MOV     DS, AX
            MOV     AX, 2508H
            INT     21H
            STI                         ; 开中断
            MOV     AX, 4C00H           ; 返回 DOS
            INT     21H                 ; 主程序结束
INTPROC     PROC                        ; 中断服务程序
            PUSH    AX                  ; 保护现场
            PUSH    BX
            PUSH    CX
            PUSH    DS
            STI                         ; 开中断
            MOV     AX, DATA            ; 在真正由外部随机信号引起的中断中，中断时数据段
            MOV     DS, AX              ; 地址 DS 是不确定的，所以程序中必须设置 DS
            INC     COUNTER             ; 中断次数加 1
            MOV     BX, OFFSET  MSG     ; 显示信息
            MOV     CX, 20              ; 20 个字符
            CALL    DISPLAY
            CLI                         ; 关中断
            MOV     AL, 20H             ; 发送 EOI 命令
            OUT     20H, AL
            POP     DS                  ; 恢复现场
            POP     CX
            POP     BX
            POP     AX
            IRET                        ; 中断返回
INTPROC     ENDP
DISPLAY     PROC                        ; 显示字符串程序
            PUSH    AX
DISPL:      MOV     AL, [BX]            ; 入口参数：BX=字符串首址
            CALL    DISPCHAR            ; CX=显示字符串个数
            INC     BX
            LOOP    DISPL
            POP     AX
            RET
DISPLAY     ENDP
DISPCHAR    PROC                        ; 显示 AL 中的字符
            PUSH    BX                  ; 入口参数：BL=字符颜色，BH=页号
            MOV     BX, 0
            MOV     AH, 0EH
            INT     10H                 ; 调用 ROM BIOS 功能显示 AL 中的字符
            POP     BX
            RET
DISPCHAR    ENDP
CODE        ENDS                        ; 代码段结束
            END     START               ; 从 START 标号处开始执行程序
```

【例 6-7】 采用时间片轮转方法实现实模式下的多任务调度。

電子工業出版社

假设有 A、B、C 三个独立任务：A 任务 4ms 执行一次，B 任务 4ms 执行一次，C 任务 2ms 执行一次，任务内容可自行设计，单个任务的程序段应在 1ms 内执行完毕。时间片长度为 1ms，由外部定时器 8253 每隔 1ms 产生中断请求信号，连接至 8259 的 IRQ_2。

定时中断调度任务在中断服务程序中实现，在中断服务程序中进行任务的调度与切换，首先以分支结构判断执行某个任务的条件是否满足，若满足则调用相应任务的子程序；否则继续判断执行下一个任务的条件是否满足。任务执行完毕后返回到中断服务程序部分，通知中断控制器中断处理结束，最后返回到主程序，等待下一次中断的来临。主程序中判断中断次数，若中断次数超过 20，则中止程序，返回 DOS。具体的主程序流程图和中断服务程序流程图如图 6-39 所示。



(a) 主程序流程图　　(b) 中断服务程序流程图

图 6-39　例 6-7 程序流程图

在时间片安排上，由于任务 A 和 B 是每隔 4ms 执行一次，任务 C 是每隔 2ms 执行一次，所以可按图 6-40 所示的方式，给任务 A、B、C 分配时间片，以达到多任务调度与切换的目的。

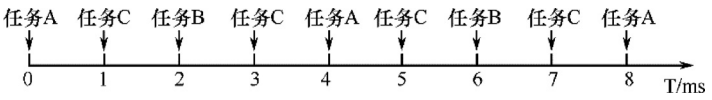


图 6-40　例 6-7 任务安排图

习 题 6

6-1　参看图 6-8，若采用无条件传送方式，编程实现以下操作：若 K_0 键单独按下，发光二极管 LED_0～LED_7 将依次点亮，每个维持 200ms；若 K_1 键单独按下，发光二极管 LED_0～LED_7 将反向依次点亮，每个也维持 200ms；其他情况下 LED 不点亮（设延时 200ms 的子程序 DELAY 已知、可直接调用）。

6-2　参看图 6-9，试编写一段程序，用查询方式从外设取 100 个数据，顺序存放在从 DATAS 开始的单元内。

電子工業出版社

6-3　已知 CPU 与打印机的连接如图 6-11 所示，打印机收到一个数据时，BUSY 信号变高，打印完以后，BUSY 信号自动变低。试编写一段程序，用查询方式将内存中从 STRING 开始的一个字符串输出到打印机，字符串结束标志为回车符。回车符也要输出。

6-4　设某输入设备，其数据端口和状态端口的地址均为 FFE0H，在读取状态时，当标志位 D_0=1 时，表示输入设备数据准备好。试编写采用查询方式进行数据传送的程序段，要求从该设备读取 100 字节，并写入到从 2000H:2000H 开始的内存中。

6-5　在 8088 系统中，从中断申请到中断服务，现场哪些部分由系统自动进行保护？哪些需由用户来保护？8088 的中断返回指令 IRET 和子程序返回指令 RET 的操作内容有何不同？

6-6　已知 SP=0100H，SS=0300H，PSW=0240H，00020H～00023H 单元的内容分别是 40H、00H、00H、01H。同时还已知 INT 8 指令存放在段地址为 0900H，偏移量为 00A0H 的 CS 段内。试指出在执行 INT 8 指令并进入该指令相应的中断例程时 SP、SS、IP、CS、PSW 和堆栈最上面三个字的内容，请画图表示。

6-7　试编写一个基于软件排优方式的中断程序，该程序控制 8 台设备，假定状态寄存器地址为 0040H，其最高位优先级最高，依次往下优先级递减；某位被置 1，表示相应设备有服务请求。同时假定已定义一个地址数组 ADDRTAB，数组中第 *i* 个元素提供第 *i* 台设备的中断服务程序入口地址的偏移量。

6-8　若 80x86 系统正以单步方式运行某用户程序（该程序已开放外部中断，即 IF=1）的过程中执行一条除法指令，INTR 线上出现了可屏蔽中断请求，与此同时，这条除法指令也产生了除法出错中断。试以流程图表示 CPU 处理这三种同时出现的中断的过程。

6-9　填空：某时刻 8259A 的 IRR 内容是 08H，说明_____。某时刻 8259A 的 ISR 内容是 08H，说明_____。在两片 8259A 级联的中断电路中，主片的第 5 级 IR_5 作为从片的中断请求输入，则初始化主、从片时，ICW3 的控制字分别是_____和_____。

6-10　若写入 8259A 的 ICW1 的命令字为 13H，则该 8259A 芯片的工作方式是什么？若写入 ICW4 为 09H，则该 8259A 芯片的工作方式是什么？

6-11　8259A 仅占用两个 I/O 地址，它是如何区别 4 条 ICW 命令和 3 条 OCW 命令的？在地址引脚 A_0=1 时，读出的是什么？

6-12　某 8086 CPU 系统中，采用一片 8259A 进行中断管理。设定 8259A 工作在普通全嵌套方式，发送 EOI 命令结束中断，采用边沿触发方式请求中断，IR_0 对应的中断类型码为 90H。假设 8259A 在系统中的 I/O 地址是 0FFDCH（A_0=0）和 0FFDDH（A_0=1）。请编写 8259A 的初始化程序段。

6-13　某 80x86 微机系统的外部可屏蔽中断由一片 8259A 管理。现要求 CPU 采用中断方式把内存从首地址 AREA 开始的长 100 字节的数据块输出到端口地址为 40H 的外设接口上。设中断类型码为 0AH，外设接口产生的中断请求信号接至 8259A 的 IR_2 引脚。试编写完成上述功能的中断服务程序及相应的主程序段，并画出简要的硬件连接图。

6-14　某 80x86 微机的中断系统有 5 个外部中断源，接在 8259A 的 IR_3～IR_7 端，中断类型码为 5BH、5CH、5DH、5EH、5FH，8259A 的端口地址为 B0H、B1H。允许它们以全嵌套方式工作，中断请求采用电平触发方式。试编写 8259A 的初始化程序。

6-15　试编写一段使 8259A 优先级顺序从高到低依次为 IR_5→IR_6→IR_7→IR_0→IR_1→IR_2→IR_3→IR_4 的程序，假定 CPU 为 8088，8259A 的偶地址为 20H，奇地址为 21H，且当前最高优先级为 IR_3。

6-16　XT 系统执行下面两条指令后，会产生什么控制状态？

電子工業出版社

```
        MOV     AL，0BCH
        OUT     21H，AL
```

6-17　编程实现将 XT 系统中 8259A 中的 IRR、ISR、IMR 内容传送至某存储区中。

6-18　自己定义一个软中断，中断类型码为 90H，在中断服务程序中检查当前数据段所在的 64K 字节内存是否能够正确地进行读写操作，方法为先向每个字节写入数据 01010101B（或 10101010B），然后读出进行比较。若正确则返回 DOS，否则调用出错处理子程序 ERROR。

6-19　如图 6-37 所示的 PC/XT 机中 8259A，其 IRQ_0 中断请求（对应中断类型码 08H）来自定时器 8253 的通道 0。系统中设置它每隔 55ms 产生一次中断，并提供完成日时钟计时功能的中断服务程序。要求替换系统计时程序，利用该中断源每隔 550ms 完成屏幕上显示一串信息："This is a 8259A interrupt！"，显示 10 次后中止，程序返回 DOS。

6-20　某 80x86 系统中设置三片 8259A 级联使用，一片为主片，两片为从片，从片分别接入主片的 IR_2 和 IR_4。若已知当前主 8259A 和从 8259A 的 IR_5 上各接有一个外部中断源，中断类型码分别为 75H、85H、95H。假设它们的中断入口地址均在同一段中，段基址为 4310H，偏移地址分别为 1230H、2340H、3450H；所有中断都采用边沿触发方式、全嵌套方式、普通 EOI 结束方式。

（1）试画出该系统的硬件连线图。

（2）试编写全部初始化程序。

6-21　在 PC/AT 中，采用两片 8259A，如图 6-38 所示。

（1）如果外部中断采用边沿信号触发、普通 EOI 结束方式，主片 IR_7（并口 1）的中断类型码为 0FH，从片 IR_1（改向中断）的中断类型码为 71H，主/从片无缓冲，写出主/从 8259A 的初始化程序。

（2）不允许串口 2（主片 IR_3）、并口 2（主片 IR_5）、软盘（主片 IR_6）、协处理器（从片 IR_5）中断，写出有关程序段。

（3）将主片和从片中的 IRR，ISR、IMR 寄存器的内容分别读到 DS:[1000]和 ES:[2000]开始的存储区域，并判断当连接主片的外部中断源有两个以上请求中断且从片只有一个中断请求时，结束从片对应的中断，并屏蔽与主片连接的所有没有请求中断的中断源。写出相应的程序段。

（4）如果在从片的 IR_7（即 IRQ_{15}）处接一个数据采集卡的中断请求端，当有中断请求时，CPU 去读取采集卡的端口（输入数据地址为 3C0H，输出数据口为 3C1H），并将一个 16 位的数据写入内存缓冲区（由 DATABUF 指示缓冲区首地址），如果读出的数据超过 2005H，则向数据输出端口发送一个命令使采集卡停止工作，命令字为 55H；如果读出的数据为 0，则向 DATABUF+5 单元写入 AAH。写出中断服务程序。

（5）如果上述数据采集卡有中断请求，使 CPU 自动执行中断服务程序，写出有关程序段。

6-22　若某 PC/AT 应用系统有 5 个故障源 A、B、C、D、E，当任一故障源变为低电平时，则有故障要处理。它们的优先权顺序是从 A～E 依次降低，处理程序的入口地址分别为 8000H、8100H、8200H、8300H、8400H。现只有 IRQ_2 可用，试按查询式中断识别与判优方案设计该中断系统，包括硬件设计和软件编程。

6-23　下面是 XT 系统 ROM　BIOS 中的 08 号中断服务程序，请说明各指令的作用。

```
INT08H  PROC    FAR
        STI
        PUSH    DS
        PUSH    AX
        PUSH    DX
```


電子工業出版社

```
        …                           ; 日时钟计时
        …                           ; 控制软驱马达
        INT     1CH
        MOV     AL, 20H
        OUT     20H, AL
        POP     DX
        POP     AX
        POP     DS
        IRET
INT08H  ENDP
```

6-24　微机的 08 号中断服务程序中，有一条调用 1CH 中断指令（INT　1CH），但系统的 1CH 中断服务程序只有一个中断返回指令。请编写一段显示信息的 1CH 中断服务程序，并希望这样中断 10 次以后恢复原来的中断服务程序。

6-25　PC 系列机的 1CH 号中断每隔 55 ms 被调用一次，它是内部中断还是外部中断？

電子工業出版社

第 7 章　微型机接口技术

7.1　接口技术概述

输入/输出设备作为计算机系统的重要组成部分，能够实现计算机与外部世界的信息交换和操作控制。这种信息交换和操作控制必须利用输入/输出设备，后者通过 I/O 接口（Interface）电路与系统相连来实现。输入/输出接口是 CPU 连接外部设备的必经通道，因此常常形象地被称为 I/O 端口（Port）。第 6 章介绍了输入/输出的基本原理、简单数字量接口和典型的输入/输出方法、中断输入及可编程中断控制器 8259。随着大规模、超大规模集成电路技术的发展，各种专用或通用接口芯片应运而生，大大降低了接口电路和信息处理的复杂程度，为微机应用的普及和发展提供了硬件基础，因此有必要系统地学习和掌握典型的接口技术。

除了地址译码和 I/O 设备选择、数据的缓冲及锁存、信息的输入与输出基本功能，典型的计算机接口技术需要解决信息的高速、高效处理与转换。

在功能设计上，当 CPU 与 I/O 设备进行数据传送时，由于 I/O 设备种类很多，可能出现 CPU 与 I/O 设备的信息类型（数字量、模拟量等）、电平（TTL 电平、RS-232C 电平等）及码制（二进制、十进制等）和信息格式（并行、串行等）等不一致的情况，此时接口必须对信息进行一定的转换以满足相应的要求。

事实上，输入/输出接口除了提供一般的逻辑信号转换，更需要解决复杂的模拟信号、多个数字系统之间各种通信编码信息的协调传送问题，尤其是要求可靠地传送连续数据信息以及长线数字信号时更复杂。图 7-1 是典型的复杂 I/O 接口原理图，其基本功能是在系统总线和 I/O 设备之间进行信号转换，满足两个时序系统之间的时间约束并提供缓冲功能。

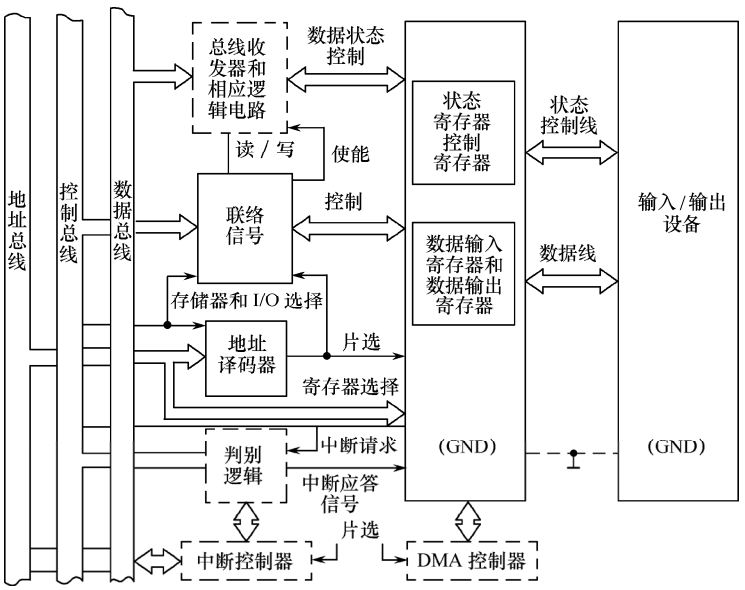


图 7-1　典型的复杂 I/O 接口原理

電子工業出版社

有时接口在设计上需要为数据传送过程中的各种工作状态提供信息，如异常或出错等，以便 CPU 进行相应处理，或支持中断、DMA 传输方式，与外部设备进行数据交换时，常常需要配置相应的握手联络（Handshacking）线来控制数据的同步交换过程。为了发挥软件的优势，面向微机应用的接口电路往往可以通过程序指令修改芯片和 I/O 引脚的工作方式，这就是所谓的可编程特性。

本章主要讨论可编程的数字量定时/计数器、并行和串行数据交换、直接存储器访问（DMA）以及实现模/数和数/模转换的模拟量典型接口电路和接口芯片的原理和应用。

7.2 可编程定时/计数器

定时/计数技术在计算机应用中具有重要作用，可实现一类时间/事件信息的输入/输出。例如在微机测控系统中，常用计数器对外部事件计数或计时，即记录来自外设的或标准的脉冲个数；需要按一定的采样周期对处理对象进行采样或检测处理，实现定时中断、检测、扫描等定时处理功能。例如，IBM PC 系列微机用一个实时时钟实现基本计时功能，还要按一定时间间隔对动态存储器进行刷新和用可变定时信号来驱动扬声器的发声等。在实时多任务操作系统中，可以利用定时器产生定时中断服务，实现进程、线程或任务调度。

实现定时或延时控制有 3 种基本方法：软件定时、不可编程硬件定时、可编程硬件定时。

（1）软件定时

利用 CPU 执行一段专门的延时程序。由于执行每条指令都需要确定的时间（指令周期），执行一个程序段可开销特定的时间，软件定时通过改变执行的指令和循环次数来控制定时时间。这种软件定时方式，简单易用，但计时不够准确，尤其是 CPU 内部有多个单元并行处理时更加复杂，难以计算。另外，软件计时独占了 CPU，降低了 CPU 的利用率。

（2）不可编程的硬件定时器

不可编程的硬件定时器分为模拟电路和数字电路两类。

① 不可编程的模拟电路硬件定时器。可由中小规模器件（如 NE555）外接电阻和电容等模拟定时元器件构成。这种方式电路结构简单，通过改变电阻和电容可使定时在一定范围内改变，由于模拟器件参数粗略，易随温度变化，定时精度一般。

② 数字电路硬件定时器。利用数字脉冲计数器，如 74LS90 十进制计数器（异步二进制加 5 进位）、92（异步二进制加六进位/十二进制）、93（加八/十六进制）、CD4040（12 级分频）等分频计数定时，这种定时电路在硬件连接好以后，定时值不易用软件来控制和改变。

（3）可编程的定时器电路

所谓可编程定时器电路，就是其工作方式、定时值、定时范围和过程可以方便地由软件来确定和改变。一个可编程定时/计数器的典型用途可包括：

① 以均匀分布的时间间隔请求中断，构成分时操作系统，以便切换任务程序（输出脉冲事件）。

② 向 I/O 设备输出精确的定时信号，该信号的周期可由程序改变。

③ 用作可编程波特率或速率发生器（可变速率），产生均匀脉冲信号。

④ 检测外部事件发生的频率或周期。

⑤ 统计外部变化过程中某一事件发生的次数。

⑥ 在定时或计数达到编程规定的值以后，产生输出信号，包括向 CPU 申请中断服务。

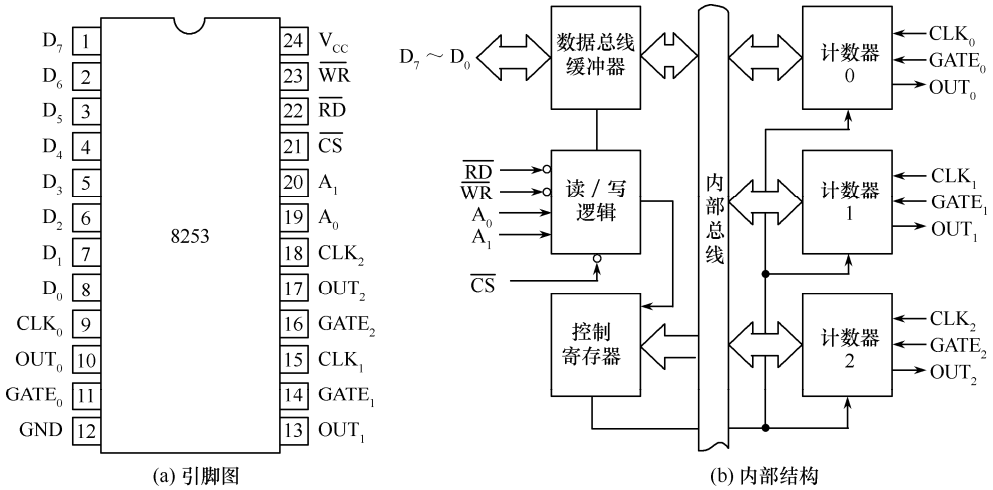
電子工業出版社

7.2.1 可编程定时/计数器 8253

8253 可编程定时/计数器是专为 Intel CPU 系统设计的接口芯片，内有三个独立的 16 位计数器（通道），工作方式可编程控制，计数脉冲频率 0～2.6 MHz（8253-5 最大计数频率 5 MHz），可以按二进制或 BCD 码减法计数，使用单一的+5 V 电源。

1．8253 芯片的内部结构和功能

8253 芯片的引脚及内部结构如图 7-2 所示，主要由数据总线缓冲器、读/写逻辑、控制寄存器及三个独立的、功能相同的计数器 0、1、2 组成。



(a) 引脚图　　(b) 内部结构

图 7-2　8253 芯片引脚及内部结构

① 数据总线缓冲器。8 位三态、双向缓冲器，用于将 8253 芯片与系统数据总线连接，CPU 写入 8253 控制字、装入计数初值或读出当前计数值。

② 读/写逻辑。接收来自系统总线的信息，产生控制信号。片选 $\overline{CS}$ 信号可允许或禁止读/写逻辑工作。

③ 控制寄存器。控制每个计数器的工作方式，包括计数方式选择及每个计数器装入初值方式。8253 控制寄存器只能写入，不能读出。

④ 三个独立计数器 0、1、2 的结构相同，各有一个 16 位可预置减法计数器（二进制或 BCD 计数），时钟输入 CLK、门控输入 GATE 和输出 OUT 引脚。输入、选通和输出由方式选择字控制，且各计数器的工作方式和工作过程完全独立。

2．8253 芯片的引脚功能

除了电源和地线，8253 芯片的主要引脚信号还有：

- D_7～D_0：双向三态数据线。
- CLK_0～CLK_2：计数器 0～2 的计数输入，要求加在 CLK 引脚的时钟周期大于 380 ns。
- $GATE_0$～$GATE_2$：门控输入，当 GATE 引脚为低电平时，禁止计数器工作；高电平时，允许计数器工作。
- OUT_0～OUT_2：计数器输出，其波形取决于工作方式。
- A_1、A_0：寻址三个计数器和控制寄存器（三个计数器的控制寄存器共用一个公共端口地址）。
- $\overline{RD}$、$\overline{WR}$、$\overline{CS}$：分别为读信号、写信号和片选信号，均为低电平有效。

電子工業出版社

A_0、A_1、$\overline{RD}$、$\overline{WR}$、$\overline{CS}$的组合可对 8253 芯片各计数器进行各种读/写操作，如表 7-1 所示。

表 7-1　8253 芯片的端口选择

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	寄存器选择及其操作
0	1	0	0	0	对计数器 0 置计数初值
0	1	0	0	1	对计数器 1 置计数初值
0	1	0	1	0	对计数器 2 置计数初值
0	1	0	1	1	对控制寄存器设置命令字或控制字
0	0	1	0	0	从计数器 0 读出计数值
0	0	1	0	1	从计数器 1 读出计数值
0	0	1	1	0	从计数器 2 读出计数值
0	0	1	1	1	无操作（D_7～D_0高阻）
1	×	×	×	×	未选中（D_7～D_0高阻）
0	1	1	×	×	无操作（D_7～D_0高阻）

3．8253 芯片的工作方式

对可编程接口芯片来说，要使其工作，首先要写入控制字或命令字，也称为对其初始化。8253 芯片没有复位引脚，在软件初始化之前，其工作方式、计数值和计数器输出状态都是不定的。

（1）控制字（Control Word，CW）

8253 芯片的控制字格式如图 7-3 所示，其中 D_5D_4=00 对应的“锁存计数器的数据”操作是为了避免 16 位计数器在高、低 8 位两次读/写中间发生翻转（借位）而提供的锁存功能。CPU 读取某通道当前的 16 位计数值前，先向控制寄存器发出锁存该通道命令，计数值即被送入相应锁存器锁存，而计数器继续工作。

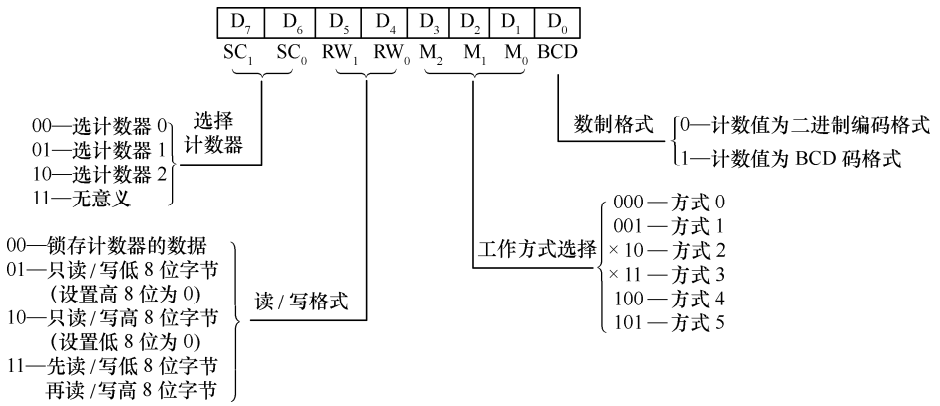


图 7-3　8253 芯片控制字格式

若选择二进制计数，则其初值范围为 0000H～FFFFH，选 0000H 时计数值最大，为 2^{16}=65536，进行二进制减法计数；若选 BCD 码计数，则初值范围为 0000～9999，其中 0000 为最大值，对应 10000，这时按 BCD 码进行减法计数。两种数制的最小值均为 1。

写入控制字后，所有控制逻辑电路复位，输出端 OUT 进入初始状态。

注意，CPU 向 8253 芯片写入的计数初值，要在 CLK 端输入一个正脉冲（一个上升沿，

電子工業出版社

然后一个下降沿）后才能被真正装入指定通道（若在此 CLK 下降沿之前读计数器，则其值是不定的）。之后再次输入时钟脉冲（CLK）才开始计数，且每次在脉冲的下降沿减 1 计数，因此实际计数值要比初值多 1。

（2）工作方式

① 方式 0——计数结束输出信号

当控制字被写入控制寄存器，OUT 输出端立即输出低电平，赋初值后，OUT 保持低电平，计数器计数，当计数减到 0 时，才变为高电平并保持，直至写入新的控制字或初值。计数器在到 0 后仍继续计数。

要实现计数，必须使 GATE 信号保持高电平。图 7-4 为方式 0 的典型计数波形，m、n 表示写入的计数初值。实际应用时，常将计数结束时 OUT 信号的上跳沿作为中断请求信号。

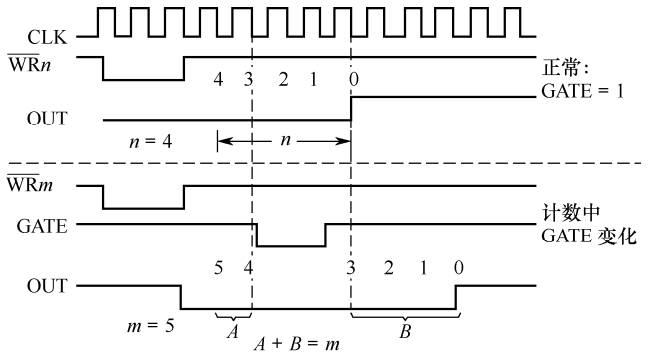


图 7-4　8253 芯片方式 0 波形

方式 0 的工作特点如下：

- 当计数到 0 后，继续减计数，OUT 输出端保持高电平。
- 在计数过程中，如果 GATE=0，则暂停计数，直到 GATE 变高后再接着计数，波形见图 7-4。
- 在计数过程中可改变计数值。若是 8 位计数，在写入新的计数值后，计数器按新值开始计数；若是 16 位计数，在写入第 1 字节后，计数器停止计数，等写入第 2 字节后，按新的初值计数。

② 方式 1——可重复触发的单稳态触发器

当 CPU 写入控制字后，OUT 输出高电平。继续写完计数初值后，计数器并不立即开始计数，直至 GATE 端到来一个上升沿（触发沿）后，在下一个 CLK 脉冲的下降沿才开始计数，OUT 变低并在计数到 0 前一直维持低电平；计数到 0 后，OUT 变高。因此输出一个单稳脉冲，如图 7-5 上半部分所示。若外部再次触发启动，则再次产生一个单稳脉冲（如图 7-5 下半部分）。方式 1 的主要特点如下：

- 可重复触发。当计数到 0 后，不需再次送计数初值，只要给它新的触发脉冲，即可产生一个同样宽度的单稳脉冲。
- 在计数过程中，若装入新的计数初值，则当前输出不受影响。只有再次触发后，计数才开始按新值输出脉冲宽度。
- 在计数过程中，外部的 GATE 触发沿提前到来，则在下一个 CLK 脉冲的下降沿，计数器开始重新计数，这将使输出单稳脉冲比原先设定的计数值加宽，如图 7-5 下半部分所示，相当于可重触发单稳电路。

電子工業出版社

③ 方式 2——频率发生器（n 分频器）

CPU 写入控制字后，OUT 变高。写入计数初值后，计数器从下一个 CLK 的下降沿开始计数。计数到 1 时，OUT 变低，经过一个 CLK 周期，OUT 恢复为高，且计数器自动重新开始计数，波形如图 7-6 上半部分所示。

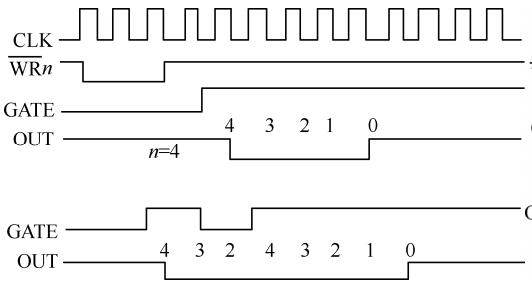


图 7-5　8253 芯片方式 1 波形

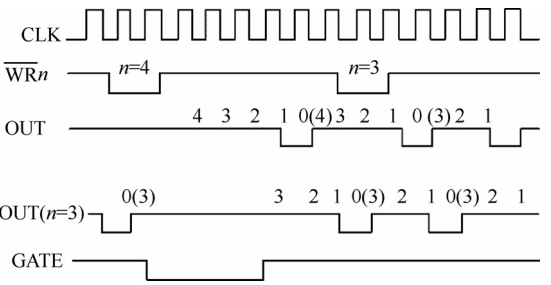


图 7-6　8253 芯片方式 2 波形

方式 2 的主要特点如下：

- 不用重复置数。计数器能够连续工作，输出 OUT 是输入 CLK 的 n（初值）分频。
- 计数过程可由 GATE 信号控制。GATE 变低将使计数暂停且 OUT 立即为高；在 GATE 变高后的下一个 CLK 下降沿计数器重新从初值开始计数，见图 7-6。这样，通过门控信号可实现计数器的同步，称为硬件同步。
- 计数器写入控制字和计数初值后，如 GATE 一直处于高电平，那么在下一个脉冲开始时计数器计数。这种通过写入计数初值使计数器同步的方式，称为软件同步。
- 计数过程中可改变初值。如果 GATE 一直处于高电平，则重装计数初值不影响当前的计数过程；计数完成后的下一个过程，才按新值计数，如图 7-6 下半部分所示。

④ 方式 3——方波发生器

方式 3 与方式 2 的工作方式类似，唯一不同的是，当计数到初值的一半时，OUT 输出变为低，直至计数到 0，输出又变高，并重新开始计数，波形如图 7-7 上半部分。

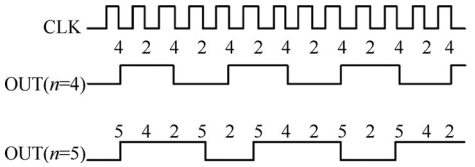


图 7-7　8253 芯片方式 3 波形

当计数初值 n 为偶数时，每个 CLK 脉冲使计数值减 2，当计数到 0 时，一方面使 OUT 输出改变状态，另一方面又重新装入计数值，开始新的计数，如此周而复始地进行重复计数；当 n 为奇数时，则第一个 CLK 脉冲使计数值减 1，之后每个 CLK 脉冲使计数值减 2，当计数到 0 时，一方面使输出改变状态，另一方面又重新装入计数值。下一个 CLK 脉冲使计数值减 3，之后每个 CLK 脉冲使计数值减 2，直至减到 0。这时，输出波形不完全对称，其中 $(n+1)/2$ 个 CLK 脉冲周期为高电平，而 $(n-1)/2$ 个脉冲周期为低电平。

⑤ 方式 4——软件触发的选通信号发生器

当写入控制字后，OUT 输出高电平。写入计数值后立即开始计数（相当于软件启动），当计数到 0 后，输出变低，经过一个 CLK 脉冲后，又变为高电平，计数器停止计数，波形如图 7-8 所示。此方式的特点是只计数一次。

GATE=1 时，允许计数；GATE=0 时，禁止计数（如图 7-8 下半部分所示）。在计数过程中改变计数初值，则按新计数值重新开始计数。如果是 16 位计数，则在写入第 1 字节时停止计数，等写入第 2 字节后，才按照新计数值开始计数。

電子工業出版社

⑥ 方式 5——硬件触发的选通信号发生器

写入控制字后，OUT 输出高电平。写入计数初值后并不立即开始计数，而是由 GATE 信号的脉冲上升沿触发启动。当计数到 0 时，输出一个 CLK 周期宽度的负脉冲，然后输出变高，停止计数。继续开始计数需等到下一次 GATE 脉冲触发，如图 7-9 所示。在计数过程中，外部 GATE 触发沿提前到来，则在下一个 CLK 脉冲下降沿计数器开始重新计数，如图 7-9 下半部分所示。

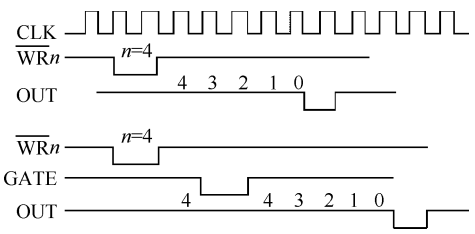


图 7-8　8253 芯片方式 4 波形

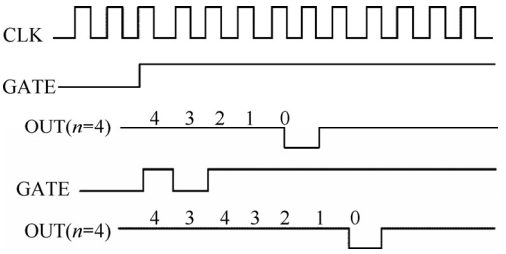


图 7-9　8253 芯片方式 5 波形

（3）8253 芯片工作方式小结

8253 芯片支持 6 种工作方式，作为二进制/BCD 码减法计数器，支持不同的触发启动方式和输出特性：方式 0，软件触发，到 0 输出电平，不自动重复；方式 1，硬件（GATE）触发单稳，不自动重复启动；方式 2、3，软硬触发，自动重装（分频/方波）；方式 4，软件触发，输出选通信号（负脉冲）；方式 5，硬件触发，输出选通信号（负脉冲）。

4．8253 芯片应用举例

（1）初始化编程

初始化编程包括两个步骤：写入控制字和写入计数初值。当 GATE=1 时，如不需要外部触发就可以工作了。例如，对计数器 0 初始化，如图 7-10 所示，使其工作于方式 1，16 位二进制计数，计数初值为 5080H，则初始化编程如下：

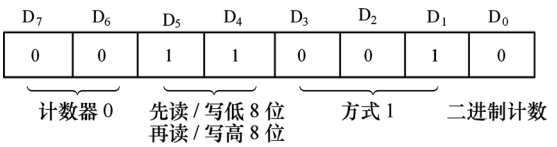


图 7-10　8253 方式 1 初始化字

① 确定控制字为 00110010B=32H。

② 计数值低 8 位为 80H，高 8 位为 50H。

设端口地址为 2A0H～2A3H，初始化程序如下：

```
MOV     DX, 2A3H                ; 控制端口地址为 2A3H
MOV     AL, 00110010B
OUT     DX, AL                  ; 送方式控制字到控制寄存器
MOV     DX, 2A0H                ; 计数器 0 端口地址为 2A0H
MOV     AL, 80H
OUT     DX, AL                  ; 先写低 8 位计数值到计数器 0
MOV     AL, 50H
OUT     DX, AL                  ; 再写高 8 位计数值到计数器 0
```

（2）读取 8253 的计数值

计数过程中，可随时读取计数器中的计数值。当选用 16 位计数方式时，CPU 要分两次读入。为了保证读到稳定准确的数据（两次读取期间计数器低位向高位无借位翻转），通常利用写入命令控制字（SC1 SC0 0 0 0 0 0 0 B）将当前计数值锁存至锁存器。锁存操作不影响计数过程。注意，控制字仍写入控制端口。

電子工業出版社

例如，要读取计数器 0 的 16 位计数值，采用锁存器锁存方式，其程序如下：

```
MOV     AL, 00H
MOV     DX, 2A3H        ; 锁存命令控制字送控制端口
OUT     DX, AL
MOV     DX, 2A0H        ; 读取计数器 0 低 8 位
IN      AL, DX
XCHG    AL, AH          ; 暂存 AH
IN      AL, DX          ; 读取读数器 0 高 8 位数据
XCHG    AL, AH          ; AX 中为计数器 0 的 16 位计数值
```

（3）应用举例

【例 7-1】 电机数字控制系统测速子系统实例。

硬件电路如图 7-11 所示。转速传感器输出的转速信号经过分压、滤波和光电耦合处理，再经过整形电路送到 8253 计数器，进行计数测量，获取转速信息。

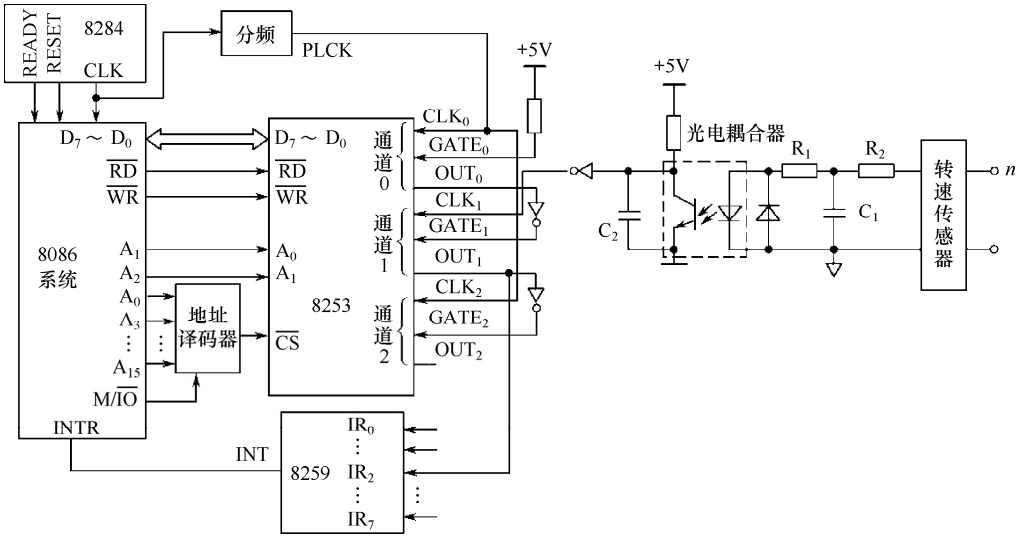


图 7-11　转速测量 8253 电路应用举例

被测电机转速 n 为 0～11156 转/分，发动机每转对应转速传感器输出 300 个脉冲。基本测量原理如下：测量 D 个（D 值根据当时实际转速范围选择）传感器输出脉冲 CLK_1 所占用的时间为多少个（设为 X 个，待测）标准时钟周期 PCLK（设为 T_r，单位 μs，由系统时钟适当分频而成），从而算出实际转速：

$$\frac{\frac{D}{300}}{\frac{n}{60}} = X \times T_r \times 10^{-6} \quad \text{即} \quad n = \frac{2 \times D \times 10^5}{X \times T_r} \text{（转/分）}$$

8253 芯片的三个计数器分别工作于方式 0、方式 1 和方式 0。图 7-12 为测量过程的时序，工作过程如下：通过 OUT_0 的启动脉冲下降沿（反相后为上升沿）触发 $GATE_1$，使计数器 1 开始对被测转速脉冲计数；同时 OUT_1 经反相后使 $GATE_2$ 为高电平（在计数期间），允许通道 2 对基准

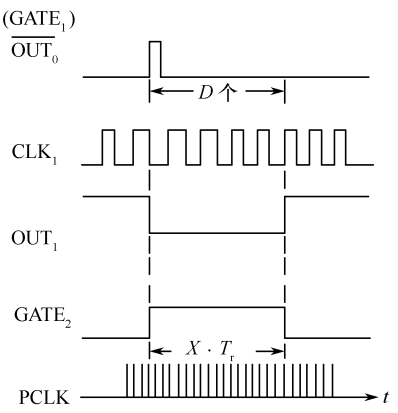


图 7-12　8253 测量应用过程时序

时钟信号 PCLK（周期 T_r）计数。在计数器 1 计满 D 个脉冲后，OUT_1 为单稳翻转，经反相后输出低电平，迫使计数器 2 停止计数，同时通过 8259 芯片的 IR_2 请求中断服务，CPU 读出计数器 2 中的计数值即可算出被测信号的转速。

设 8253 端口地址为 280H、282H、284H、286H（思考：为什么），主要程序段如下：

```
; 主程序中对 8253 初始化
INIT_C:  MOV     DX, 286H          ; 控制端口地址为 286H
         MOV     AL, 52H           ; 01010010B，计数器 1 初始化：方式 1，读/写低字节
         OUT     DX, AL            ; 送方式控制字到控制寄存器
         MOV     AL, 0B0H          ; 10110000B，计数器 2：方式 0，读/写双字节
         OUT     DX, AL
         MOV     AL, 10H           ; 00010000B，计数器 0：方式 0，读/写低字节
         OUT     DX, AL
INIT_D:  MOV     DX,282H           ; 计数器 1
         MOV     AL, 20
         OUT     DX, AL            ; 写入计数初值 D=20 到计数器 1
         INC     DX
         INC     DX                ; 计数器 2
         MOV     AL, 0
         OUT     DX, AL            ; 写入低 8 位计数值到计数器 2
         OUT     DX, AL            ; 写入高 8 位计数到计数器 2，初值 0 相当于 10000H
         MOV     DX, 280H          ; 计数器 0
         MOV     AL, 01H
         OUT     DX, AL            ; 写入计数初值 1 到计数器 0、启动测量过程
         …
; 定时中断服务程序中，读取计数器 2 的数据，计算转速部分
IN_DATA: MOV     AL, 80H           ; 10000000B
         MOV     DX, 286H
         OUT     DX, AL            ; 锁存计数器 2
         MOV     DX, 284H          ; 读计数器 2 的内容 10000H-X
         IN      AL, DX
         MOV     AH, AL
         IN      AL, DX
         XCHG    AL, AH
         NEG     AX                ; AX=X
         …                         ; 计算转速 n
```

7.2.2 可编程定时/计数器 8254

8254 是 8253 的超集（即 8254 向下兼容 8253），包括 8253 的所有功能，引脚排列与 8253 完全相同，还增强了以下功能：① 计数脉冲频率范围扩大，最高可达 10 MHz（8254-2）。② 增加了状态“读回（Read-Back）”读命令，允许用户检查计数初值、编程方式、OUT 引脚当前状态以及清除所选计数器的计数标志。

8254 芯片允许一条读回命令（写入控制端口）锁存最多全部三个计数器的当前计数值和状态信息，其命令格式如图 7-13 所示（在 8253 中，这是无效命令）。单个对多个计数器的锁

電子工業出版社

A_0A_1=11，　$\overline{CS}$=0，　$\overline{RD}$ =1，　$\overline{WR}$ =0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT_2	CNT_1	CNT_0	0

D_5=0：锁存所选计数器的计数值；D_4=0 锁存所选计数器的状态；D_3=1：选择计数器 2

D_2=1：选择计数器 1；D_1=1：选择计数器 0；　D_0：保留将来扩充用，必须为 0

图 7-13　8254 读回命令格式

存操作命令等效于一组对单计数器的操作命令。对某个计数器执行读操作后对应计数器自动解除锁存，但其他计数器仍保持锁存直到完成读操作。例如，读命令 C2H=11000010B 为读回计数器 0 的计数值和状态，结果是锁存计数器 0 的计数值和状态；而读命令 E4H=11100100B 为读回计数器 1 的状态，结果是锁存计数器 1 的状态。

当读回命令把某通道的计数值和状态信息都锁存时，对该端口第一次读操作读取的是状态字节，其位 D_5～D_0 与初始化方式字完全相同，OUTPUT 位 D_7 含 OUT 引脚的当前状态，由此允许用户通过软件方法监测 OUT 引脚的状态，进而有可能简化系统的部分硬件。接下来读取的 1 或 2 字节（取决于该通道工作方式的定义）为锁存的计数值。

应用中对多个计数器的读回命令产生的锁存操作和软件读操作有一定的顺序要求。

7.3　可编程并行接口

一个字符的 *n* 个数位用 *n* 条线同时传输的机制被称为并行通信。并行通信具有传输速度快、效率高等优点，常用于数据传输速度要求高而传输距离较短的场合。并行通信由并行接口完成，前面介绍的三态缓冲器 74LS244、锁存器 74LS273 等都是简单的并行接口芯片。

7.3.1　可编程并行接口芯片 8255A

Intel 8255A 是可编程通用并行接口芯片，它有 24 只功能可编程的 I/O 引脚，与 Intel 系列 CPU 完全兼容，能直接按位清 0 或置 1，简化了控制应用接口。

1．8255A 芯片的内部结构

8255A 芯片内部结构如图 7-14 所示，它由以下 4 部分组成。

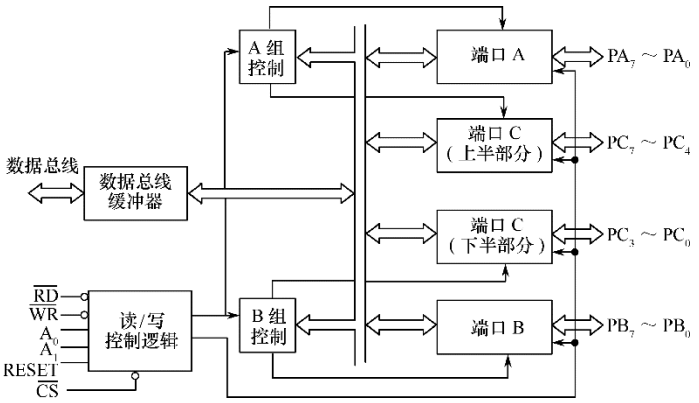


图 7-14　8255A 芯片内部结构

① 三个 8 位数据端口。

電子工業出版社

端口 A：内有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入锁存器。

端口 B：内有一个 8 位数据输入/输出、锁存/缓冲器和一个 8 位数据输入缓冲器。

端口 C：内有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入缓冲器（输入无锁存），可分为两个 4 位端口使用，或用作与 A 口或 B 口配合的控制或状态口，依具体工作方式而定。

② A 组控制和 B 组控制。端口 A 和端口 C 的高 4 位构成 A 组，端口 B 和端口 C 的低 4 位构成 B 组，分别由 A 组和 B 组控制电路控制。两组控制电路内各有一个控制寄存器，接收 CPU 写入的控制字，决定各端口的工作方式。

③ 双向 8 位数据总线缓冲器。承担与 CPU 数据总线接口的功能，传送 I/O 数据、CPU 控制字及状态信息。

④ 读/写控制逻辑。接收 CPU 发出的地址（一般为 A_1、A_0）及控制（$\overline{RD}$、$\overline{WR}$、RESET）和片选（$\overline{CS}$）信号，产生给 A 组、B 组的控制信号，以完成对数据、状态及控制信息的传送。

2．8255A 芯片引脚功能

8255A 芯片引脚定义如图 7-15 所示。除了电源和地线，其他引脚信号可以分为两类。

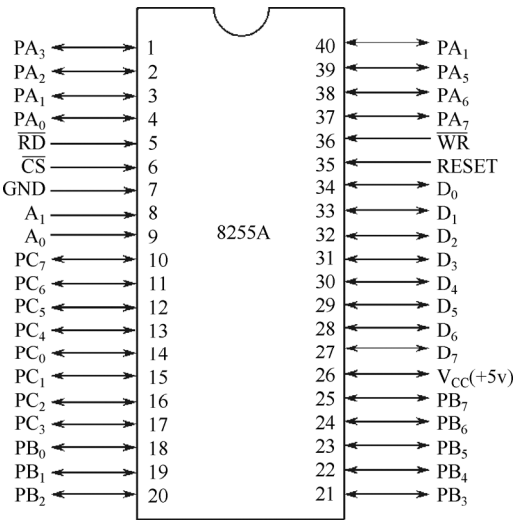


图 7-15　8255A 芯片的引脚定义

（1）与外设相连的引脚

PA_7～PA_0：端口 A 数据线。

PB_7～PB_0：端口 B 数据线。

PC_7～PC_0：端口 C 数据线。

（2）与 CPU 相连的引脚

RESET：复位信号，高电平有效。复位时，所有内部寄存器均被清 0，三个数据端口被设为输入方式。

D_7～D_0：数据总线，双向，三态。

$\overline{CS}$：片选信号，低电平有效。

$\overline{RD}$：读信号，低电平有效。

$\overline{WR}$：写信号，低电平有效。

A_1、A_0：端口选择信号。用来寻址三个数据端口及一个控制端口。

電子工業出版社

A_1、A_0、$\overline{RD}$、$\overline{WR}$ 和 $\overline{CS}$ 组合，完成 8255A 芯片的基本操作，如表 7-2 所示。

表 7-2　8255A 芯片端口选择操作

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	端口选择及其操作
0	1	0	0	0	数据送端口 A
0	1	0	0	1	数据送端口 B
0	1	0	1	0	数据送端口 C
0	1	0	1	1	控制字送控制寄存器
0	0	1	0	0	端口 A 数据送数据总线
0	0	1	0	1	端口 B 数据送数据总线
0	0	1	1	0	端口 C 数据送数据总线
0	0	1	1	1	无操作（D_7～D_0 三态）
1	×	×	×	×	禁止（D_7～D_0 三态）
0	1	1	×	×	无操作（D_7～D_0 三态）

3．8255A 芯片的控制字和工作方式

8255A 芯片有三种基本工作方式。可通过向控制寄存器写入控制字来选择工作方式。

（1）8255A 的控制字

控制字分为两类，共用一个地址，由 D_7 的值来区分。

① 方式选择控制字，如图 7-16 所示。D_7=1 是其特征标志位。A 组有三种工作方式：方式 0，基本输入/输出方式；方式 1，选通输入/输出方式；方式 2，双向传输方式。B 组只能工作于方式 0 或方式 1。

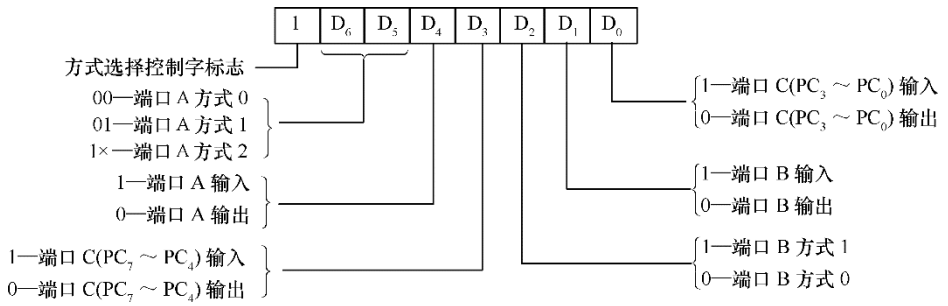

1 D6 D5 D4 D3 D2 D1 D0
方式选择控制字标志
00—端口 A 方式 0
01—端口 A 方式 1
1×—端口 A 方式 2
1—端口 A 输入
0—端口 A 输出
1—端口 C(PC7 ～ PC4) 输入
0—端口 C(PC7 ～ PC4) 输出
1—端口 C(PC3 ～ PC0) 输入
0—端口 C(PC3 ～ PC0) 输出
1—端口 B 输入
0—端口 B 输出
1—端口 B 方式 1
0—端口 B 方式 0

图 7-16　8255A 方式选择控制字

② 端口 C 置位/复位控制字，如图 7-17 所示，支持对外设的按位控制。控制字的 D_3～D_1 指明对端口 C 的哪一位进行操作，而 D_0 位则指明是置 1 还是清 0，D_6～D_4 位为任意值。

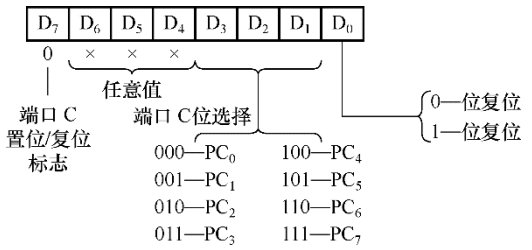


图 7-17　8255A 端口 C 置位/复位控制字

【例 7-2】 使端口 C 的 PC_7=1，则控制字为 00001111B，即 0FH；然后使 PC_3=0，则控制字为 00000110B，即 06H。设 8255A 控制端口地址为 286H，程序段如下：

電子工業出版社

```
        MOV     AL, 0FH             ; 置 PC7=1 的控制字
        MOV     DX, 286H            ; 控制端口地址
        OUT     DX, AL              ; 置 PC7=1
        MOV     AL, 06H             ; 置 PC3=0 的控制字
        OUT     DX, AL              ; 置 PC3=0
```

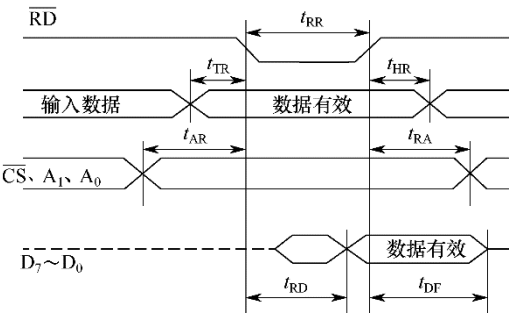
（2）8255A 芯片的工作方式

① 方式 0——基本输入/输出方式

在这种方式下，三个数据端口 A、B、C（C 分为两个 4 位），通过方式选择控制字可任意选择为输入口或输出口。方式 0 的主要特点如下：

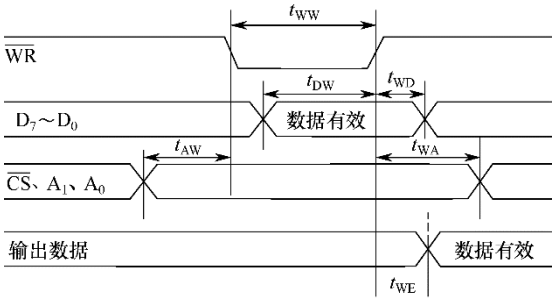
- 两个 8 位端口 A、B 和两个 4 位端口（端口 C），任一个端口都可以作为输入或输出端口，各端口之间没有规定必然的关系，有 16 种可能的输入/输出组合。
- 输出锁存，而输入不锁存。

方式 0 的输入/输出时序如图 7-18 和图 7-19 所示。方式 0 主要用于同步传送数据的场合，这时 CPU 和外设相互了解对方的工作状态，不需要应答信号，三个数据端口可实现三个 8 位通道的数据传送（图 7-18 中表列 t_{Ry} 为两次读操作间隔，图中未画出）。



参数	说　明	8255A	
		最小时间	最大时间
t_{RR}	读脉冲的宽度	300 ns	
t_{AR}	地址稳定领先于读信号的时间	0	
t_{TR}	输入数据领先于 $\overline{RD}$ 的时间	0	
t_{HR}	读信号过后数据继续保持时间	0	
t_{RA}	读信号无效后地址保持时间	0	
t_{RD}	从读信号有效到数据稳定的时间		250 ns
t_{DF}	读信号撤除后数据保持时间	10 ns	150 ns
t_{RY}	两次读操作之间的时间间隔	850 ns	

图 7-18　8255A 方式 0 输入时序及说明



参数	说　明	8255A	
		最小时间	最大时间
t_{WW}	地址稳定领先于写信号的时间	0	
t_{AW}	写脉冲的宽度	400 ns	
t_{DW}	数据有效时间	100 ns	
t_{WD}	数据保持时间	30 ns	
t_{WA}	写信号撤除后的地址保持时间	20 ns	
t_{WE}	写信号结束到数据有效的时间		50 ns

图 7-19　8255A 方式 0 输出时序及说明

方式 0 也可用于查询式传送场合。这时，一个数据端口作为状态/控制口，另两个数据端口为输入/输出口，利用状态/控制口来配合数据输入/输出口的操作。例如，设端口 A、B 为数据口，端口 C 的高 4 位为控制输出口，低 4 位为状态输入口，则使端口 C 与端口 A、B 配合，即可以实现查询式传送。

② 方式 1——选通的输入/输出方式

在这种方式下，端口 A 和 B 输入/输出数据时，必须利用端口 C 的选通信号和应答信号（握手信号），而端口 C 的这些位有一定的约定。方式 1 的主要特点如下：

電子工業出版社

- 两组端口（A和B）都可工作于方式1。每组包含一个8位数据端口和一个4位控制/数据端口。
- 8位数据口可以是输入/输出，输入/输出均带锁存。
- 4位端口用作8位端口的控制/状态位。未用于控制/状态的位仍可用作基本I/O。

方式1输入时，如图7-20所示，其各控制信号的含义如下。

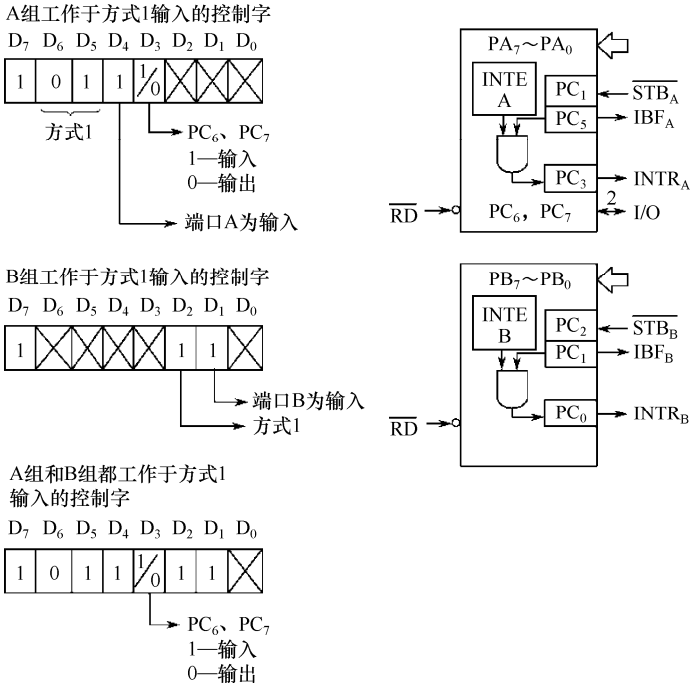


图7-20　8255A方式1输入时端口对应的控制信号

$\overline{\text{STB}}$：选通输入，低电平有效，是外设提供的输入信号。当其有效时，把外设送来的数据送入8255A的输入锁存器。

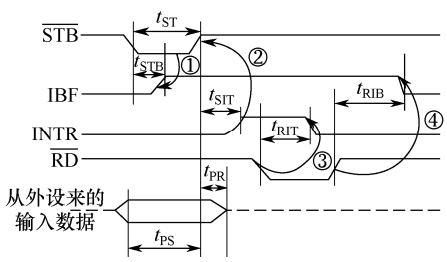
IBF：输入缓冲器满，高电平有效，是8255A输出到外设的联络信号，有效时表示数据已写入锁存器，即写入缓冲器满，以通知外设暂停送数。

INTR：中断请求信号。高电平有效。是8255A的输出信号，用于向CPU提出中断申请，请求CPU读取数据。INTR在$\overline{\text{STB}}$、IBF以及INTE（中断允许）都为1时被置1（有效）。

INTE：中断允许信号，由置位/复位控制字对PC_4（用于端口A）或PC_2（用于端口B）位置1或清0来控制。置1时，允许中断；清0时，禁止中断。注意，这里的INTE是8255内部寄存器位，与PC引脚位地址相同，但不受PC引脚状态影响。

8255A芯片工作于方式1的输入时序如图7-21所示。当外设的数据已送到8255A的端口（A或B），并用选通信号$\overline{\text{STB}}$把数据锁存到8255A的输入锁存器时，经过t_{STB}时间后，IBF信号有效。它既可通知外设，暂不输入新的数据，又可供CPU查询。在选通信号结束t_{SIT}时间后，8255A可向CPU发出INTR中断请求。如果中断允许的话，CPU响应中断，把数据读入CPU。在$\overline{\text{RD}}$信号有效t_{RIT}时间后撤销中断请求。$\overline{\text{RD}}$信号结束t_{RIB}时间后，IBF变低，表示写入缓冲器已空，通知外设可写入新的数据。

電子工業出版社



参数	说　明	8255A	
		最小时间	最大时间
t_{ST}	选通脉冲的宽度	500 ns	
t_{STB}	选通脉冲有效到 IBF 有效之间的时间		300 ns
t_{SIT}	$\overline{STB}$ =1 到中断请求 INTR 有效之间的时间		300 ns
t_{PR}	数据保持时间	180 ns	
t_{PS}	数据有效到 $\overline{STB}$ 无效之间的时间	0	
t_{RIT}	$\overline{RD}$ 有效到中断请求信号撤除之间的时间		400 ns
t_{RIB}	$\overline{RD}$ 为 1 到 IBF 为 0 之间的时间		300 ns

图 7-21　8255A 方式 1 的输入时序

方式 1 输出时的控制信号如图 7-22 所示，其各控制信号的含义如下。

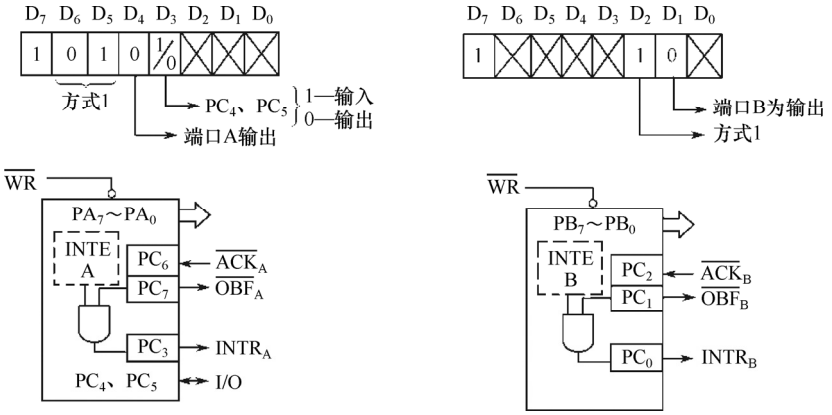


图 7-22　8255A 方式 1 输出时端口对应的控制信号

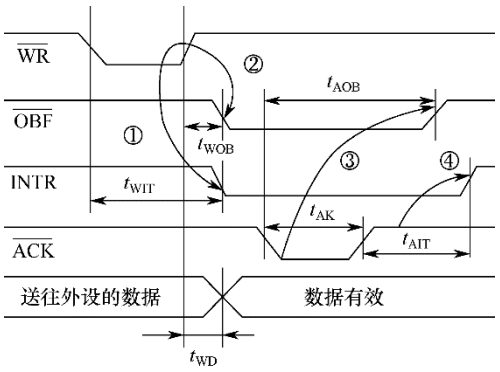
$\overline{OBF}$：输出缓冲器满，低电平有效，由 8255A 输出给外设。当其有效时，表示外设可从指定端口读取 CPU 写入的数据。

$\overline{ACK}$：外设应答信号，低电平有效。有效时，表示外设已经从 8255A 输出端口读取了数据。

INTR：中断请求信号，高电平有效，输出缓冲区空，请求 CPU 向 8255A 指定端口写入数据。其有效条件为：$\overline{ACK}$ =1，$\overline{OBF}$ =1 及 INTE=1。

INTE：中断允许。由 PC_6（端口 A）或 PC_2（端口 B）的置位/复位控制。

8255A 工作于方式 1 输出的时序如图 7-23 所示。



参数	说　明	最小时间	最大时间
t_{WIT}	从写信号有效到中断请求无效的时间		850 ns
t_{WOB}	从写信号无效到输出缓冲器清空的时间		650 ns
t_{AOB}	$\overline{ACK}$ 有效到 $\overline{OBF}$ 无效的时间		350 ns
t_{AK}	$\overline{ACK}$ 脉冲的宽度	300	
t_{AIT}	$\overline{ACK}$ 为 1 到发新的中断请求的时间		350 ns
t_{WD}	写信号撤除到数据有效时间		350 ns

图 7-23　8255A 方式 1 输出的时序及说明

采用中断控制方式工作时，8255A 的输出过程在中断服务程序中完成，CPU 通过 OUT

電子工業出版社

指令输出数据和（I/O）写信号，$\overline{WR}$ 信号一方面清除 INTR，另一方面使 $\overline{OBF}$ 有效，通知外设接收数据（实际上，$\overline{OBF}$ 信号可以是给外设的一个选通信号）。当外设接收数据后，发出 $\overline{ACK}$ 信号，它一方面使 $\overline{OBF}$ 失效，表示数据已取走，当前输出锁存器空；另一方面在其上升沿使 INTR 有效，即发出新的中断请求，形成一个新的输出过程。

方式 1 的状态信号可通过读取端口 C 得到，其格式如图 7-24 所示。

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
输入方式	I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$
输出方式	$\overline{OBF_A}$	$INTE_A$	I/O	I/O	$INTR_A$	$INTE_B$	$\overline{OBF_B}$	$INTR_B$
	A 组					B 组		

图 7-24　8255A 方式 1 的状态信号格式

因此，采用方式 1 异步传送数据，CPU 既可工作于中断方式，也可工作于查询方式。

③ 方式 2——双向传输方式

此方式只适合端口 A，这时在 PA_7～PA_0 的 8 位数据线上，外设既可从 8255A 芯片获取数据，也可向 8255A 芯片发送数据，并支持查询和中断方式。其主要特点是：一个双向 8 位总线端口（A）和一个 5 位控制/状态端口（C）；输入/输出均带锁存。

方式 2 的控制和状态信息如图 7-25 所示，示例时序如图 7-26 所示。

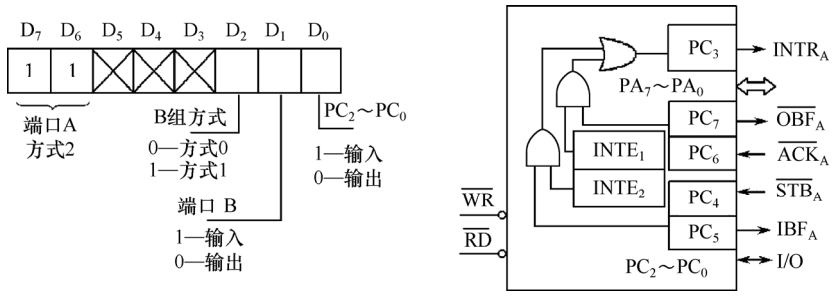


图 7-25　8255A 方式 2 的控制和状态信号

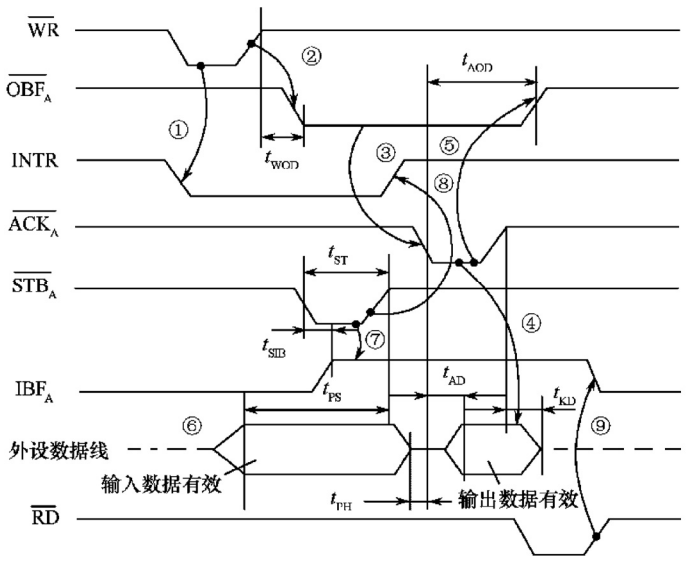


图 7-26　8255A 方式 2 示例时序

8255A 方式 2 的控制信号功能类似方式 1。其中 $INTE_1$ 和 $INTE_2$ 分别对应于方式 2 的输出中断允许 $INTE_A$（PC_6）和输入中断允许 $INTE_B$（PC_4）。$INTR_A$（PC_3）为输入和输出的中断请求信号，高电平有效。其有效条件为

$$INTR=IBF\cdot INTE_2\cdot \overline{STB}\cdot \overline{RD}+\overline{OBF}\cdot INTE_1\cdot \overline{ACK}\cdot \overline{WR}$$

方式 2 的时序实质上是方式 1 的输入和输出方式的组合，故各个时间参数的意义、数值也相同。需要说明的是，图 7-26 时序是示例，输入、输出顺序是任意的。

方式 2 的状态字格式如图 7-27 所示，可通过读端口 C 得到。各端口各种允许的工作方式和输入/输出方式可任意组合。端口 C 未用作控制/状态的位，可根据输入/输出定义，用端口 C 读和复位/置位控制字实现输入/输出操作。

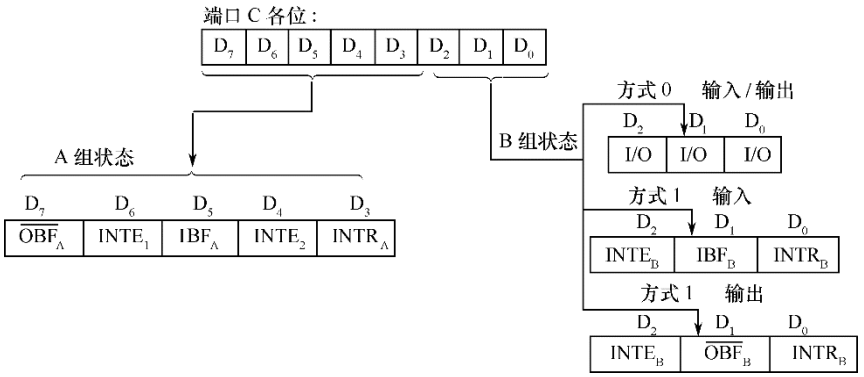


图 7-27　8255A 方式 2 时端口 C 的各种组态

4．8255A 芯片初始化与 I/O 应用举例

【例 7-3】 设 8255A 工作于方式 0，端口 A 为输入，端口 B 为输出，端口 C 为输出，试对其进行初始化。

首先确定方式控制字为 10010000B，设 8255A 端口地址为 80H～83H。初始化程序如下：

```
MOV     AL，90H              ；方式选择控制字 10010000B
OUT     83H，AL              ；方式选择控制字送 8255A 控制端口
```

写完控制字后，CPU 可通过 IN/OUT 指令来与 8255A 芯片传送数据。例如：

```
IN      AL，80H              ；读端口 A 的数据
…
OUT     81H，AL              ；AL 中数据写入端口 B
…
OUT     82H，AL              ；AL 内容写入端口 C
```

7.3.2　并行打印机接口应用

1．打印机的主要接口信号与时序

以点阵式微型打印机 TPμP-40P 为例，打印机接收标准 ASCII 字符和控制码，并转换成字模点阵打印输出。

D_7～D_0：数据总线，双向、三态。

$\overline{STB}$：数据选通触发脉冲，打印机在其上升沿读入数据。

$\overline{ACK}$：应答脉冲，“低”表示数据已被打印机接收且打印机准备好接收下一数据，常用

電子工業出版社

于打印机的中断申请信号。

BUSY：“高”表示打印机正“忙”，不能接收数据，通常用作状态信号输入供 CPU 查询。其他还有在线、出错、缺纸等状态信号。并行打印机的基本时序如图 7-28 所示。

2．查询方式打印字符串

CPU 通过 8255A 芯片与打印机的基本接口如图 7-29 所示。8255A 的端口 A 作为数据通道，工作在方式 0、输出方式；由 PC_7 读入 BUSY 状态，PC_0 输出 STB 脉冲，故 C 端口也工作在方式 0，上半部输入、下半部输出；B 端口未用。

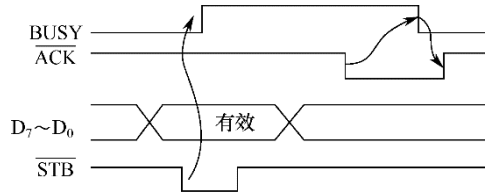


图 7-28　并行打印机基本时序

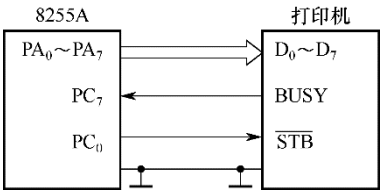


图 7-29　8255A 与并行打印机接口原理图

设 8255 端口地址为 280H～283H。打印子程序（过程）如下：

```
BUF     DB      'HELLO!'
        DB      0DH                     ; 回车符 ASCII 值
        DB      0AH                     ; 换行符
NUM     EQU     $-BUF                   ; 打印字符串长度
        …
PRINT   PROC  FAR
        MOV     DX, 283H
        ;8255 初始化:均为方式 0,
        MOV     AL, 10001000B           ; A 口输出，C 口上半部输入、下半部输出
        OUT     DX, AL
        MOV     AL, 00000001B           ; 控制口，初始 STB  PC0=1
        OUT     DX, AL
        MOV     SI, OFFSET BUF
        MOV     CX, NUM
NEXT:   MOV     DX, 282H
        IN      AL, DX                  ; BUSY=1（忙）?
        TEST    AL, 80H
        JNZ     NEXT                    ; PC7=0，不忙
        MOV     AL, [SI]
        INC     SI
        MOV     DX, 280H
        OUT     DX, AL                  ; 送出数据
        MOV     DX, 283H
        MOV     AL, 00000000B
        OUT     DX, AL                  ; STB  PC0=0（低电平宽度≥0.5μs）
        NOP
        MOV     AL, 00000001B
        OUT     DX, AL                  ; 恢复 STB 高电平 PC0==1
        LOOP    NEXT
```


電子工業出版社

```
        RET
PRINT   ENDP
```

3．中断方式打印字符串

采用中断方式打印时，8255A 芯片与打印机的基本接口如图 7-30 所示。8255A 的端口 A 作为数据通道，工作在方式 1、输出方式。PC_6 作为 $\overline{ACK}$ 信号输入端，而 PC_3 作为 INTR 信号输出端，连接至 PC 机 8259 的中断请求信号输入端 IRQ_2（中断类型码 0AH，对应的中断向量放在 0000:0028H～0000:002BH 单元中）。打印机需要的数据选通（$\overline{STB}$）信号由 CPU 控制 PC_0 来产生。这时 PC_7（$\overline{OBF}$）引脚未用，故将其悬空。

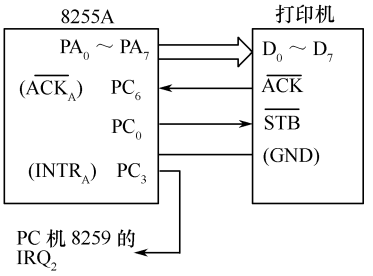


图 7-30　8255A 中断作为打印机接口原理

打印过程如下：设数据放在输出缓冲区，输出字符时，CPU 通过对 PC_0 置 1/清 0 命令产生输出数据选通脉冲，把端口 A 的数据送到打印机。当打印机接收并打印字符后，发出 $\overline{ACK}$ 应答信号，由此信号清除 8255A 的 $\overline{OBF}$ 信号（内部与 INTR 相联），并使 8255A 产生新的中断请求。如果 CPU 允许中断，则响应中断，进入中断服务子程序，输出新的字符。

在需要打印时，首先调用打印初始化子程序，对 8255A、中断向量表等进行初始化，并允许响应中断。之后 CPU 每次响应打印机中断申请，则在中断服务程序中输出一个字符到打印机。设 8255A 端口地址仍为 280H～283H。程序如下：

```
IRQ      EQU     0AH              ; IRQ2中断类型码
IMR1     EQU     0FBH             ; 11111011B
IMR2     EQU     04H              ; 00000100B
BUF      DB      'HELLO!'
         DB      0DH, 0AH         ; 回车，换行符
NUM      EQU     $-BUF
BUFPT    DW      ?                ; 保存打印缓冲区当前指针
BUF_N    DB      NUM              ; 打印字符计数器
…
…
PRT_INI  PROC  FAR                ; 初始化子程序
         CLI
         MOV     DX, 283H         ; 8255初始化:A口方式1、输出，
         MOV     AL, 10100000B    ; B口方式0，C口下半部输出
         OUT     DX, AL
         PUSH    DS
         MOV     AX, CS
         MOV     DS, AX           ; 设置中断向量
         LEA     DX, PRT_INTR
         MOV     AH, 25H
         MOV     AL, IRQ
         INT     21H
         IN      AL, 21H
         AND     AL, IMR1         ; 设置8259的中断屏蔽寄存器（IRQ2开中断）
         OUT     21H, AL
         POP     DS
         STI
```


電子工業出版社

```
        MOV     DX, 283H
        MOV     AL, 00000001B       ; 初始 STB =1
        OUT     DX, AL
        MOV     SI, OFFSET BUF
        MOV     AL, [SI]
        INC     SI
        MOV     BUFPT, SI
        MOV     BUF_N, NUM-1
        MOV     DX, 280H            ; 送出第一个数据
        OUT     DX, AL
        MOV     DX, 283H
        MOV     AL, 00000000B       ; STB =0
        OUT     DX, AL
        NOP
        MOV     AL, 00000001B       ; STB =1
        OUT     DX, AL
        RET
PRT_INI ENDP
…
PRT_INTR PROC  FAR                  ; 中断服务子程序，未考虑数据段变化
        PUSH    AX
        PUSH    DX
        PUSH    SI
        MOV     SI, BUFPT
        MOV     AL, [SI]
        INC     BUFPT               ; 送一个数据
        MOV     DX, 280H
        OUT     DX, AL
        MOV     DX, 283H
        MOV     AL, 00000000B       ; STB =0
        OUT     DX, AL
        NOP
        MOV     AL, 00000001B       ; STB =1
        OUT     DX, AL
        DEC     BUF_N               ; 字符串打印完毕?
        JNZ     NEXT
        IN      AL, 21H             ; 打印完毕
        OR      AL, IMR2            ; 中断屏蔽寄存器: IRQ2 关中断
        OUT     21H, AL
NEXT:   POP     SI
        POP     DX
        MOV     AL, 20H             ; 通知 8259: 中断服务结束
        OUT     20H, AL
        POP     AX
        IRET
PRT_INTR  ENDP
```


電子工業出版社

7.3.3 键盘和显示器接口

在计算机系统中，键盘和显示器是必不可少的 I/O 外设，连接控制方法很多，传统上多通过并行接口连接。这里只举例介绍其中最基本的非编码键盘和 LED 数码显示器的应用。

1．非编码键盘

键盘的工作方式一般有定时扫描和中断扫描两种。中断扫描方式是在有键按下（按键开关闭合）时产生中断请求，CPU 在中断服务程序中进行键盘扫描及分析处理，一般通过专用芯片（如可编程键盘/显示器控制器 8279）实现。定时扫描一般利用定时器产生定时（如每隔 20 ms）中断、CPU 响应中断后对键盘进行扫描，或软件扫描，有键按下时转入分析处理程序。

为了减少引线，键盘一般为矩阵式。图 7-31 中系统采用的是一个 4×4 键盘。键盘扫描电路可用专用或通用并行接口芯片实现，采用 8255 芯片更为方便。按键的识别通常有两种方法：行反转法和行扫描法。

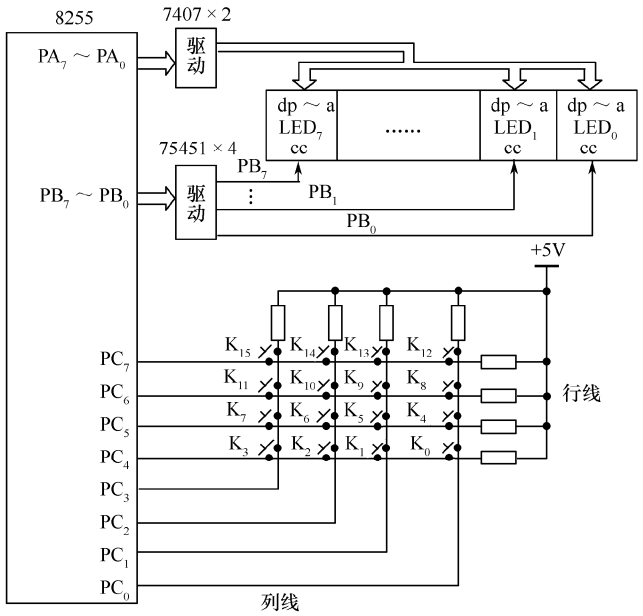


图 7-31　8255 用作键盘及 LED 显示器接口

① 行反转法。程序先令行线接口工作在输出方式而列线接口工作在输入方式，并使行线输出全 0，然后读入列线值，如果此时有某键按下，则使对应列值为 0；然后程序再令列线接口工作在输出方式而行线接口工作在输入方式，并使列线输出全 0，然后读入行线值，则闭合键对应的行值为 0。因此，当一个键被按下时，可以读到相应的一对唯一的行值和列值。由于反转法是通过行列颠倒两次扫描来识别闭合键的，因此需要两个可编程双向 I/O 端口（见图 7-31），用 8255 端口 C 的上、下半部分别用作键盘的行线和列线。

② 行扫描法。程序使某行为低电平、其余行为高电平，然后读入并查询列值。如果列值中某位为低电平，则说明行列交叉点处的键被按下；否则扫描下一行，直到扫描完全部行线。在实际应用时，一般先快速检查键盘中是否有键按下，再具体确定按下的是哪个键。为此，先使所有行输出都为 0，再检查列线输入是否有 0。如果有，此时由于不能确定闭合键所

電子工業出版社

在的行，于是用行扫描法来具体定位。如果读到的数据全部为 1，则说明无键闭合。硬件上可将 8255 端口 C 的上（输出方式）、下半部（输入方式）分别用作键盘的行线和列线，如图 7-31 所示（图中行线的上拉电阻可去掉）。

图 7-31 所示系统中完整的键盘管理程序包括键扫描、消除抖动、键译码等内容。用行扫描法实现键盘管理的子程序（现场保护略）举例如下：

```
KEY_V    DB       ?                    ; 键值暂存单元
         …
         …
KEY      PROC  FAR
START:   MOV      DX, CTRL_PORT        ; 8255A 方式控制字送控制端口 CTRL_PORT
         MOV      AL, 10000001B
         OUT      DX, AL
         CALL     KEYSCAN      ; 键扫描。有键按下时：BL=键值（0～15）；无键按下时：BL=0FFH
         CMP      BL, 0FFH
         JNZ      WAIT
         RET                           ; 无键按下，返回
WAIT:    MOV      KEY_V, BL
         CALL     DELAY                ; 延时约 2ms，消除抖动
         CALL     KEYSCAN
         CMP      BL, KEY_V
         JZ       RELEASE      ; 两次键值相同，确认该键按下，等待释放，否则认为是干扰
         RET
RELEASE: MOV      AL, 0FH              ; 往所有行线 PC7～PC4 上输出 0
         MOV      DX, PORTC            ; PORTC 为 8255A 端口 C
         OUT      DX, AL
         IN       AL, DX               ; 读所有列线 PC3～PC0 的电平状态
         AND      AL, 0FH              ; 比较，是否有列线处于高电平（键释放）
         CMP      AL, 0FH
         JNZ      RELEASE
         MOV      AL, KEY_V            ; 取键值
         MOV      AH, 0                ┐
         ADD      AL, AL               │
         MOV      BX, OFFSET PROCE     ├; 散转（键译码）
         ADD      BX, AX               │
         MOV      AX, CS:[BX]          ┘
         JMP      AX                   ; 或 CALL  AX，如果处理分支为过程
PROCE    DW       KEY0                 ┐
         DW       KEY1                 ├; 键功能处理入口地址表
         …                             ┘
         DW       KEY15                ; 键 15 处理程序

KEY0:    …                             ; 各键功能处理，Key0～Key15
         RET
KEY1:    …
         RET
KEY15:   …
         RET
```


電子工業出版社

```
KEY     ENDP
; 键盘扫描子程序（过程）
KEYSCAN PROC
        MOV     AL, 0FH             ; 往所有行线 PC7～PC4 上输出 0
        MOV     DX, PORTC           ; PORTC 为 8255A 端口 C
        OUT     DX, AL
        IN      AL, DX              ; 读所有列线 PC3～PC0 的电平状态
        AND     AL, 0FH             ; 比较，是否有列线处于低电平
        CMP     AL, 0FH
        JNZ     DONE                ; 有键按下，转 DONE
        MOV     BL, 0FFH
        RET                         ; 无键按下，返回，BL=0FFH
DONE:   MOV     BL, 0               ; 键号初值为 0
        MOV     CL, 0EFH            ; 送扫描初值 11101111B（D3～D0 无效）
        MOV     CH, 4               ; 计数值为行数
FROW:   MOV     AL, CL              ; 扫描一行
        MOV     DX, PORTC
        OUT     DX, AL
        RCL     AL, 1               ; 修改行扫描值
        MOV     CL, AL
        IN      AL, DX              ; 读列线 D7～D4 无效，判别是否有列线为低电平
        AND     AL, 0FH
        CMP     AL, 0FH
        JNZ     FCOL                ; 有列线 D3～D0 为低电平，则转 FCOL
        ADD     BL, 4               ; 没有，则使键号寄存器 BL 的值=BL+列数(4)
        DEC     CH                  ; 行扫描未完，则转 FROW
        JNZ     FROW
        RET                         ; 已扫描完，返回
FCOL:   RCR     AL, 1               } ; 此列为低电平，已找到，返回（BL=键号）
        JNC     FCOL1
        INC     BL                  } ; 未找到低电平的列线，则使键号 BL=BL+1，并继续寻找
        JMP     FCOL
FCOL1:  RET
KEYSCAN ENDP
```

在此程序中，没有考虑重键（即多个键同时按下）等问题。

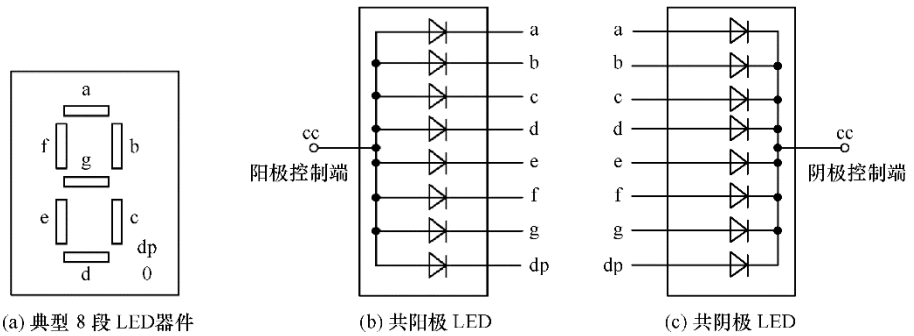
2．LED 数码显示

LED 数码显示器是一种常用的显示器，由 8 段发光管组成，如图 7-32(a)所示，它们分别称为 a、b、c、d、e、f、g、dp 段。通过 8 段的亮灭组合，可以显示 0～9 和 A～F 等字符及小数点。LED 有共阳极和共阴极两种结构，分别如图 7-32(b)和(c)所示。

当使用共阴极结构时，阴极控制端为低电平，数码显示端输入高电平（通常加限流电阻）时发亮。例如，要显示数字 1，则使 b、c 段为高电平，其他段为低电平；使用共阳极时，有效电平相反。阳极控制端为高电平，数码显示端输入低电平时发亮。8 段发光管按表 7-3 所示与数据端口连接时，数字 0～9 等的编码顺序排列，定义为一个数据表，称为“段选码表”（共阴极 1、共阳极 0 对应段亮）：

```
TAB     DB      3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH, 77H, 7CH
        DB      39H, 5EH, 79H, 71H          ; 共阴极 LED，同相输出，显示字符 0～F
```


電子工業出版社



(a) 典型 8 段 LED器件　(b) 共阳极 LED　(c) 共阴极 LED

图 7-32　8 段 LED 数码显示器

或

```
TAB     DB      0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H, 88H
        DB      83H, 0C6H, 0A1H, 86H, 8EH       ; 共阳极 LED，同相输出，显示字符 0～F
```

表 7-3　8 段发光管各段对应的数据位

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
dp	g	f	e	d	c	b	a

在多个 LED 显示电路中，通常把阴（阳）极控制端连接到一个输出端口，即位控端口；而把数码显示段连接到另一个输出端口，也称为段控端口。由于点亮 LED 需要一定的电流（每段为 5～20 mA），因此在接口中需要增加驱动电路和限流电阻，图 7-31 中使用了同相驱动器 7407（加上拉排电阻，如 510 Ω）和 75451。

在图 7-31 中，8255A 的端口 A 用来输出显示字符。设 TAB 为 LED 段选码表的首地址，那么要显示的数字的地址正好为“起始地址+数字值”，其地址中存放着对应于该数字值的显示码。例如显示 7，则在它对应的显示码 TAB+7 这个单元中，利用 80x86 换码指令 XLAT，可方便地实现数字到显示码的转换。

8255A 的端口 B 用来控制 LED 的显示位，即位控端口。在软件设计上通过扫描法逐个接通 8 位 LED，把端口 A 输出的显示码送到相应的 LED 位上显示，以减少硬件开支。这时，8255A 端口 A 送出的段选显示码尽管输出到了各 LED，但由于端口 B 只有一位输出低电平，所以只有一个 LED 的相应段能够导通而显示数字，其他 LED 并不亮。这样，端口 A 依次输出段选码，端口 B 依次选中一位 LED，就可以在各位上显示不同的数据。利用眼睛的视觉暂留特性，当采用一定的频率循环地往 8 位 LED 输送显示码和扫描码时，就可看到稳定的数字显示。这种 LED 显示方式称为动态刷新。8 位 LED 显示刷新一遍的子程序（过程）如下（其中子程序保护部分略去）：

```
TAB     DB      3FH, 06H, …, 71H        ; 0～F 的段选码表，共阴极 LED
BUF     DB      8 DUP(?)                ; 显示缓冲区
        …
        …
DISPLAY PROC  FAR
        MOV     SI, OFFSET BUF          ; 指向缓冲区首地址
        MOV     CL, 7FH                 ; 使最左边 LED 亮
DISI:   MOV     AL, [SI]                ; AL 中为要显示的字符
        MOV     BX, OFFSET TAB          ; 段选码表首址送 BX
        XLAT                            ; 段码送 AL
```


電子工業出版社

```
        MOV     DX, PORTA           ; 段码送段控端口 A: PORTA
        OUT     DX, AL
        MOV     AL, CL              ; 位扫描码送位控端口 B: PORTB
        MOV     DX, PORTB
        OUT     DX, AL
        CALL    DELAY               ; 延时 1ms
        CMP     CL, 0FEH            ; 扫描到最右边 LED?
        JZ      QUIT                ; 是，则已显示一遍，退出
        INC     SI                  ; 否，则指向下一位 LED
        ROR     CL, 1               ; 位码指向下一位
        JMP     DISI                ; 显示下一位 LED
QUIT:   RET
DISPLAY ENDP
```

轮换显示过程中，考虑到显示效果，显示下一数码应该在改变位码前先清除段码（如送 FFH 至 PORTA，即各段全暗），避免出现前一数字的“影子”。

键盘和显示器接口还可以有其他多种实现方案，如利用不可编程的并行接口芯片或专用的可编程键盘/显示控制芯片 8279 等。

7.4 串行接口与串行通信

并行传输的最大特点是多位一次传送，速度快、连线多。但许多 I/O 器件与 CPU 交换信息，或计算机与计算机之间交换信息时，往往通过一对导线或通信通道来传送信息。这时，每一时刻只传送一位信息，每位占据一个规定长度的时间段，这种数据逐位顺序传送的方式称为串行传输。与并行传输方式相比，串行传输具有传输线少、成本低的特点，特别适合远距离传送，其缺点是速度慢，若并行传送 n 位数据需要时间 T，则串行传送的时间至少需 nT。

串行传输方式主要用于各类本地或远程通信系统，并通过专门的通信接口芯片实现信息的传输及传输控制。串行传输由于应用接口连线简单，易于隔离，因此也被用于单元模块及芯片间的数据交换，如串行接口的打印机、存储器、键盘显示器及接口控制芯片等，并形成了标准的串行传输总线，如 I^2C、SPI 等。

7.4.1 串行通信的基本概念

1. 全双工、半双工和单工

全双工通信：如图 7-33(a)所示，分别用独立的接口和传输线来发送和接收信号，使发送和接收可同时进行，这种通信方式称为全双工通信。

半双工通信：数据可以两个方向传输，但不能同时双向传输，只能分时交替进行，如图 7-33(b)所示。通过软件和接口的协调控制，实现传输换向。

单工通信：只允许单方向传送数据，如图 7-33(c)所示，只能从 A 发送到 B，应用较少。

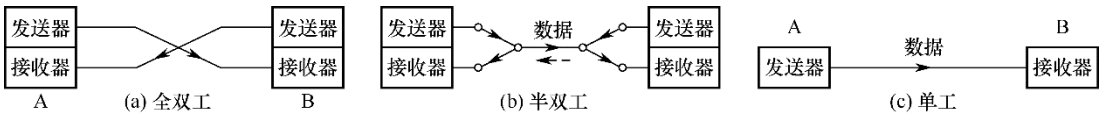


图 7-33 串行通信方式

電子工業出版社

2．异步通信与同步通信

为了保证数据通信的顺利进行，通信双方必须约定并遵守基本通信规程，即通信协议。异步通信和同步通信是两种基本的串行通信方式，实际上也是两类不同通信协议的一部分。

（1）异步数据通信

异步数据通信（Asynchronous Data Communication）是指通信中收发双方没有共同的时间基准，往往以字符为单位一位一位进行传输、接收。异步通信格式如图 7-34 所示，用一个起始位表示字符的开始，用停止位表示字符的结束，在起始位和停止位之间是 5～8 位字符位及奇偶校验位。这种由起始位表示字符的开始、停止位表示字符的结束所构成的一串数据也称为字符帧。从图 7-34 可以看到，一帧数据各位的时间间隔是固定的，而相邻两帧数据的时间间隔是不固定的。

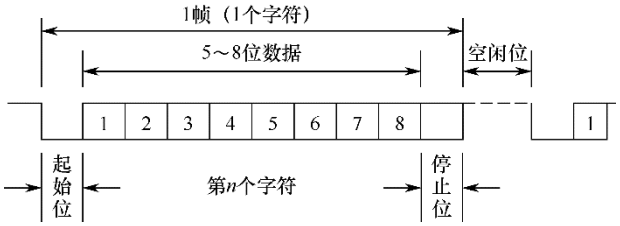


图 7-34　异步通信格式

在异步通信中，按字符一帧一帧传送，每个字符的传送靠起始位来同步，起始位固定（1 位），发送和接收之间还需约定：

① 字符的格式。即约定字符帧的编码形式，奇偶校验和停止位的个数。例如，传送 ASCII 值时，可选 7 位字符、1 位偶校验位、1 个停止位，加起始位共 10 位为一帧。

② 波特率（Baud rate）。即约定传送数据位的速度。用位/秒（b/s）来表示。例如，设数据传送的速率为 120 字符/秒，每个字符（帧）包括 10 个数据位，则不考虑空闲位，传送的波特率为

$$10\times120=1200\text{ 位/秒}=1200\text{ 波特}$$

而每一位的时间宽度为

$$T_{\mathrm{d}}=\frac{1}{1200}=0.833\,\mathrm{ms}$$

通常，异步通信的波特率为 50～9600，高速的可达 19200。在点对点串行通信中大多采用异步通信，它允许发送端和接收端有 4%～5%的时钟误差或波特率误差。

（2）同步数据通信

由于异步通信是按字符帧进行数据传送的，每传送一个字符都必须配上起始位、停止位，这就使异步通信的有效数据传送效率降低 1/4～1/5。为了提高速度，就要求取消这些辅助信息位，引入同步措施。

为了便于接收处理，同步通信在数据块开始处设置 1～2 个同步字符来进行判别，在串行数据线上，始终保持连续的字符，即使没有字符时，也要发送专用的“空闲”字符或同步字符，如图 7-35 所示。

同步数据通信（Synchronous Data Communication）速度远高于异步数据通信，工作在几十到几百千波特乃至千兆波特，一般要求用精确的同步时钟来控制发送端和接收端之间的同步，因为发送端和接收端之间的时钟信号有微弱的偏差，在长时间的通信中将产生累积误差，导致通信出错，因此同步通信的硬件比较复杂。

電子工業出版社

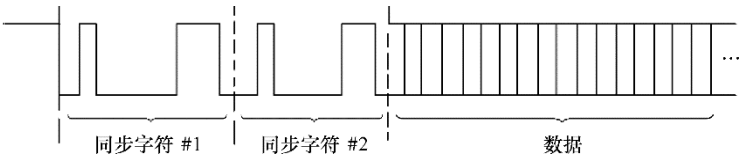


图 7-35　同步通信的一类格式

3．数据编码

在通信设备内部传输数据时，由于各电路功能模块之间的距离短，工作环境可控，传输过程中一般采用简单高效的数据信号传输方式，如直接将二进制电平信号送传输通道进行传输。但在远距离传输中，由于线路较长，数据信号在传输介质中将会存在损耗干扰，为减少在特定介质中的损耗和干扰影响，需要进行传输转换，使之成为适于在该介质上传输的信号。这一过程称为信号编码。

常用的信号编码类型有：单/双极性不归零（Non Return to Zero，NRZ）码、单/双极性归零码、差分编码、单极性相位编码（曼彻斯特码）和交替反转码等。

在归零编码中，正电平代表逻辑 1，负电平代表逻辑 0，每传输完一位数据，信号返回到 0 电平，也就是说，信号线上会出现 3 种电平：正电平、负电平、零电平。这种编码就相当于把时钟信号用归零方式编码到了数据中。这样的信号也叫自同步（Self-clocking）信号，可以省去时钟数据线，但是大部分的数据带宽都用来传输“归零”而浪费掉了。不归零编码则省去归零，对应的 NRZI 编码（Non-Return-to-Zero Inverted）用信号的翻转代表一个逻辑，信号保持不变代表另外一个逻辑。NRZ 和 NRZI 都没有自同步特性，但是应用中可以用一些特殊的技巧解决问题。

另外，常用的单极性相位编码（曼彻斯特）编码波形如图 7-36 所示，其编码规则如下：对于数据代码 1，用电平正跳变（或负跳变）来代表，数据代码 0 与数据代码 1 跳变相反，跳变在一位数据传输时间内完成。所谓正跳变指一位代码前半个周期为低电平，后半个周期为高电平，而负跳变正好相反。

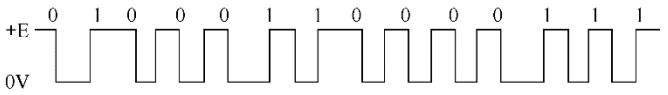


图 7-36　单极性相位编码波形

4．信号的调制/解调

计算机通信是一种数字信号通信，要求传输线的频带很宽。如长距离通信采用电话线传递信息，而电话线的频带没有这么宽，这时简单地直接使用电话线去传送数字信号，就会造成信号的畸变。

为了保证信号传送的可靠性，在长距离通信中，常采用调制解调器（Modem）来保证信号品质。调制器（Modulator）把数字信号转换为模拟信号，经过传输线传送到目的地后，再用解调器（Demodulator）检测此模拟信号，再把它转换成数字信号，如图 7-37 所示。通常把调制和解调功能做在一起，构成 Modem。串行通信中用一对 Modem 来处理信号转换。

调制/解调的实现方法很多，如移频键控法（FSK）、幅移键控法（ASK）、相移键控法（PSK）、相移幅度调制（PAM）等。FSK 是其中常用的一种，把数字信号 1 和 0 调制成不同频率的模拟信号，这两个不同频率的模拟信号，分别由电子开关控制，在运算放大器的输入端相加，而电子开关由要传输的数字信号（即数据）控制。

电子工业出版社

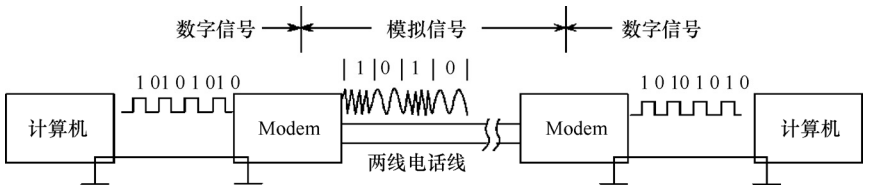


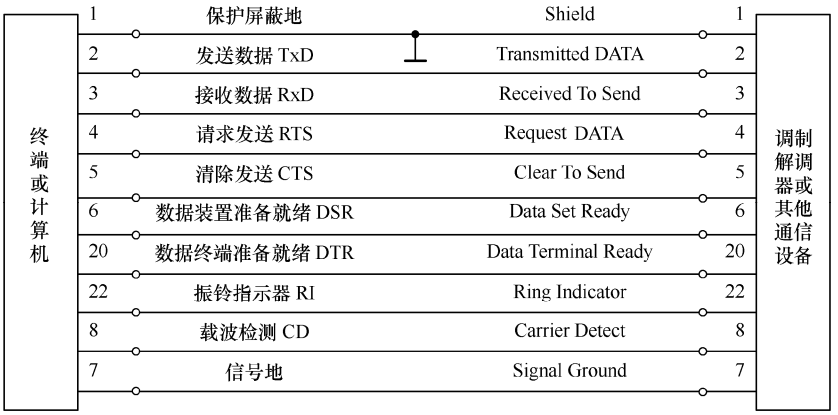
图 7-37　信号的调制与解调实例

5．RS-232C 与 RS-422/485 总线接口

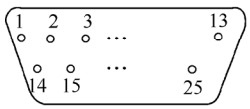
在串行通信中，考虑到长线收发需要，还需要有相应的电路，并形成一定的接口标准。常用的接口标准有 EIA RS-232C、RS-422/485 等。

（1）RS-232C 接口标准

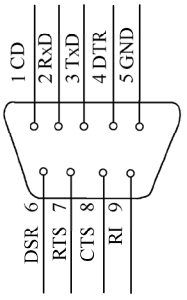
RS-232C 是串行异步通信中应用最广的串行总线标准，主要用于使用模拟信道传输数字信号的场合，推出这种标准的最初目的是在数据终端设备 DTE（Data Terminal Equipment）与数据通信设备 DCE（Data Communication Equipment）之间建立接口标准。如 PC 系统调制解调器 Modem 为 DCE 设备，而计算机的串行接口装置（UART）相当于 DTE 设备。RS-232C 采用 25 芯 D 型连接器（见图 7-38(b)），每个引脚信号和电平都有标准规定，任何具备 RS-232C 接口的设备都不需要附加其他硬件就可与计算机连接。图 7-38(a)是其基本的常用信号规定。在微机中还常用 9 芯 D 型连接器，如图 7-38(c)所示。



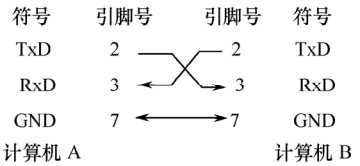
(a) 引脚信号



(b) D型 25 芯插头



(c) D型 9 芯插头



(d) 基本信号连接（25 芯标准）

图 7-38　RS-232C 信号

在串行通信中，除了收发数据线和地线，为了保证信息的可靠传送，还有若干联络控制信息线互相连接。这些联络控制线包括如下。

① 请求发送 $\overline{\text{RTS}}$（Request To Send）：当发送器已经做好发送的准备，要了解接收方是否已做好接收的准备，是否可以开始发送，就要向对方输出一个有效的 $\overline{\text{RTS}}$ 信号，以等待对方的回答。

② 准许发送 $\overline{\text{CTS}}$（Clear To Send）：当接收方做好接收的准备，在接收到发送方送来 $\overline{\text{RTS}}$ 信号后，就以有效的 $\overline{\text{CTS}}$ 信号作为应答。

③ 数据终端准备就绪 $\overline{\text{DTR}}$（Data Terminal Ready）：通常当某个设备的接收器已做好接收的准备时，为了通知发送器可以发送，就向发送器送出一个有效的 $\overline{\text{DTR}}$ 信号。

④ 数据装置准备就绪 $\overline{\text{DSR}}$（Data Set Ready）：当发送方接收到接收方送来的有效的 $\overline{\text{DTR}}$ 信号，在发送方做好发送准备后，就向接收方送出一个有效的 $\overline{\text{DSR}}$ 信号作为应答。

振铃指示器 RI 和载波检测 CD 作为 Modem 输出到接收方的信号，通知接收方准备接收数据。这两个信号通常用于电话线载波通信中。

除了对信号引脚的定义做出规定，RS-232C 还规定信号电平标准（EIA）采用负逻辑电平：–3～–25 V 为 1 电平，+3～+25 V 为 0 电平。

RS-232C 长线收发器可实现 EIA 电平与 TTL 电平的转换，典型芯片如 MC1488/1489，目前多采用单电源收发器 ICL232/MAX232 等，如图 7-39 所示，工作电源为单一+5 V，通过电荷泵变换电路获得±9 V 电源。在三线方式下（见图 7-38(d)），一个 ICL232 芯片可以实现两组串口信号的电平变换。FC232 系列则是带有光电隔离电路的 RS-232C 串行通信转发器，可以实现 4 线双向全双工通信。

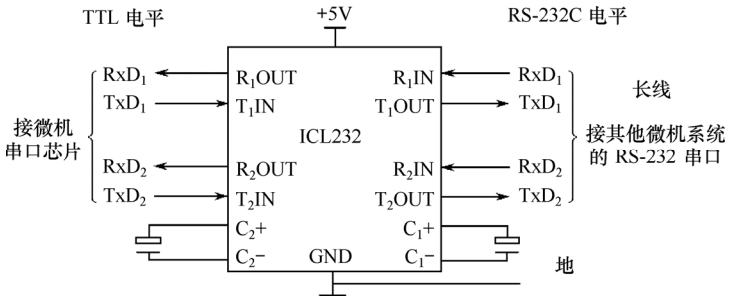


图 7-39　TTL 电平与 RS-232C 电平转换

（2）RS-422/485 接口标准

RS-232C 是目前应用广泛的一种串行异步通信接口，采用单端驱动、单端接收电路，即有公共地线，抗干扰能力差，不能进行远距离传输。

RS-422A 采用平衡驱动、差分接收电路，抗共模干扰信号能力强，其对逻辑电平的定义是根据两条传输线之间的电位差值来决定的，如当电位差值低于–0.2 V 时表示逻辑 0，高于+0.2 V 时表示 1。另外，RS-422A 的最大传输速率可达 10 Mb/s（当传输距离为 15 m 时），最大传输距离可达 1200 m（当传输速率为 90 kb/s 时）。相比之下，RS-232C 的最大传输速率为 20 kb/s，最大传输距离为 15 m。

RS-422A 的另一特点是，允许驱动器输出电压为+2～+6 V，接收器输入电平灵敏度为±0.2 V。采用 4 线传输（一对用于发送，一对用于接收），可以实现多站互联通信，但标准规定电路中只有一个发送器，可以有多达 10 个接收器。

RS-232C 和 RS-422A 用于点对点的异步串行通信，但是在许多应用环境中，要求用较少的信号线来实现通信，或者要求在同一通信网络中允许有多个发送器，这时需采用 RS-485

電子工業出版社

接口标准。RS-485 串行接口总线实际上是 RS-422A 的变形，即 RS-422A 为全双工模式，而标准 RS-485 为半双工模式，这就对实现多站（节点）互连提供了很大方便。图 7-40 为点对多点通信时，RS-485 与 RS-422A 的连接电路。该电路既可构成 RS-422A 电气接口（按图中虚线连接，即采用 4 线传输信号），也可构成 RS-485 电气接口（按图 7-40 中实线连接，即采用 2 线传输信号）。此外，在 RS-485 互连中，因是半双工方式，某一时刻只能有一个站发送数据，其他站只能接收，因此 RS-485 发送端由使能端控制，正常情况下，各站此端应为“无效”，禁止发送。只有在本站需要发送时，才将使能端变为有效，由发送端将数据发出。需要等到移位发出后，使能端关闭，以便接收对方来的数据。

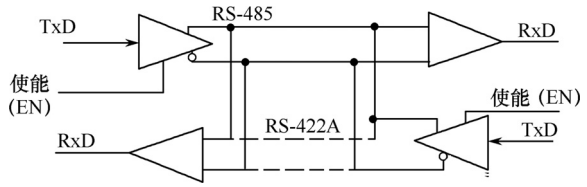


图 7-40　RS-485 与 RS-422A 的连接形式

RS-485 可实现多点对多点的半双工通信，在同一网络中的平衡电缆线上，最多可连接 32 个发送器/接收器对，加上它在抗干扰能力、最大传输距离和速率方面的优势，已被广泛用于实时控制、微机测控网络等领域。

在实际应用中，常常需要支持全双工模式的 RS-485，即将 RS-422 扩展到点到多点，采用 4 线制，可避免收/发切换控制的复杂性。

6．串行通信的实现

上面主要讨论了异步串行通信的“信道”问题，可以看出，串行传送数据是逐位依次传送的，而数据在计算机中却是并行的，为此要实现串行通信收发就必须解决串行到并行和并行到串行的转换问题，通常用硬件电路解决，如硬件 UART（Universal Asynchronous Receiver/Transmitter，通用异步接收器/发送器）。

UART 内部原理如图 7-41 所示，收/发都是双缓冲器结构。接收时，由 RxD 来的串行数据先进入移位寄存器，然后并行输入至接收缓冲器，由 CPU 从数据总线读取；发送时，从 CPU 来的并行数据由缓冲器暂存，然后送至移位寄存器，由 TxD 一位一位输出。

在 UART 工作时，检测电路始终监视着 RxD 线，当确定起始位后，就开始字符的接收过程。UART 通过外接时钟与接收数据同步，通常其时钟周期 T_c 和数据位周期 T_d 之间的关系为 $T_c = T_d / K$，式中 K=16 或 64。

设 K=16，在 RxD 线上字符开始的前沿被检测到后，接收器就以 16 倍的时钟频率来控制采样时间，以便接收器能在一个字符周期 T_d 的 1/16 时间内确定字符的开始，随后的采样全部以 16 倍频时钟为基准进行，如图 7-42 所示，其工作过程如下：

（1）当字符的前沿出现时将计数寄存器清 0。

（2）启动计数器，使 16 倍频时钟的每个脉冲对计数器加 1。

（3）当计数器连续计数达到 8 个 0 时，则确定它为起始位（不是干扰信号），并表示已达到起始位的中间位置，从此点开始，重新设置计数器，使计数器清 0。

（4）此后每隔 16 个时钟脉冲采样一次数据线，作为输入数据。计数器清 0，如此重复直至最后的停止位被采样。

（5）如果停止位是正确的，字符被接收到缓冲寄存器，然后再从第（1）步开始。

为了检测长距离传送中可能产生的错误，在 UART 中还有一些传送出错标志，以提高传送的可靠性。常见的错误有以下 3 种。

電子工業出版社

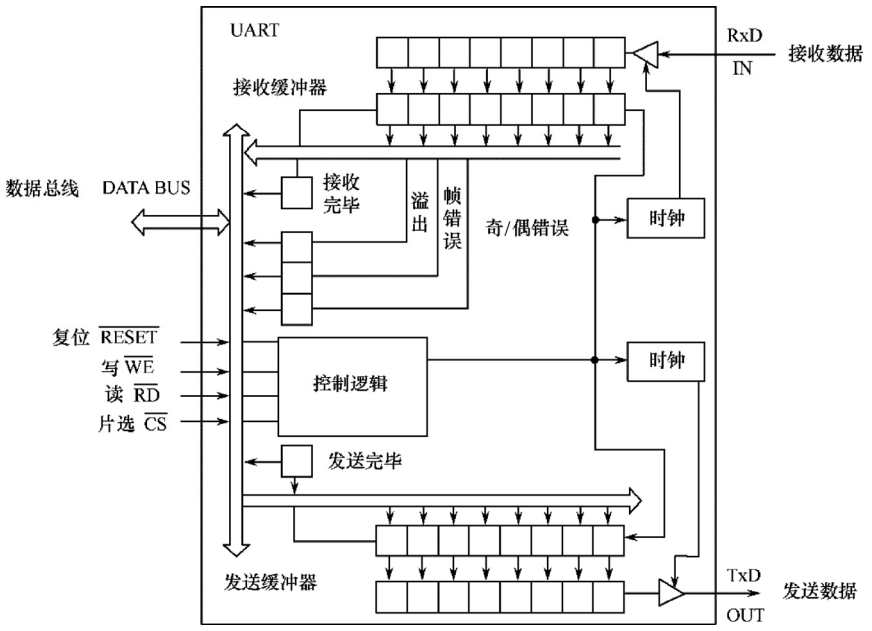


图 7-41　UART 电路原理

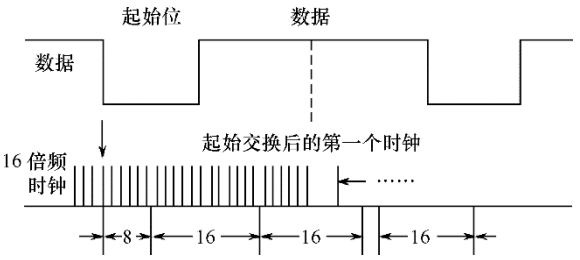


图 7-42　16 倍频时钟对字符再同步

① 奇偶校验错误（Parity Error）。接收器按照事先约定的方式（偶校验、奇校验或无奇偶校验）进行奇/偶校验计算，即接收到的字符中 1 的个数与约定（期望）不符，置此标志发出奇偶出错信息。

② 帧错误（Frame Error）。接收到的字符不符合规定，如缺少停止位等，则置此标志发出帧出错信息。

③ 溢出（丢失）错误（Overrun Error）。在接收到新的字符后，由串行移位寄存器传送到并行接收寄存器，如果此时并行接收寄存器中的字符还未被 CPU 读走，出现数据丢失现象；反之发送寄存器数据还没有来得及送到移位寄存器，又有新的字符写入，也属于溢出错误。

串行通信通常使用专用控制接口芯片简化管理控制，常用的可编程通信接口芯片有 Intel 8251A 和个人计算机常用的 Intersil INS 8250（NSC NS16450）等。

7.4.2　可编程串行通信接口 8251A

1．Intel 8251A 芯片的性能与工作原理

（1）基本功能

8251A 是一种可编程串行通信接口芯片，通过编程可以使之工作在异步或同步方式。

電子工業出版社

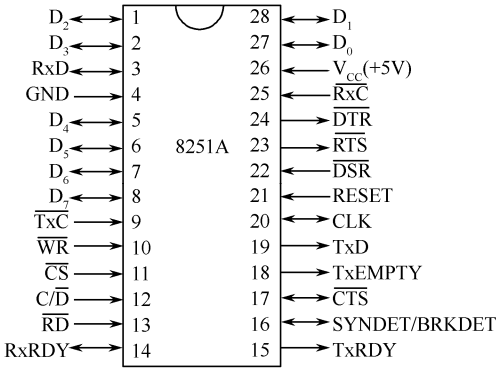


图 7-43　Intel 8251A 芯片引脚信号

8251A 芯片采用 28 脚 DIP 封装，如图 7-43 所示。它支持全双工、双缓冲收发。在同步方式下，波特率最高为 64 kb/s，传输格式为 5～8 位数据位，内部能自动检测同步字符，允许有奇/偶校验位。异步方式下，功能比 INS8250 芯片简单，波特率最高为 19.2 kb/s，传输格式为 1 位起始位、1 位奇/偶校验以及 5～8 位数据位和 1、1.5、2 位停止位。

（2）8251A 芯片结构与工作原理

8251A 由发送器、接收器、数据总线缓冲器、读/写控制电路及调制解调控制电路等 5 部分组成，如图 7-44 所示。

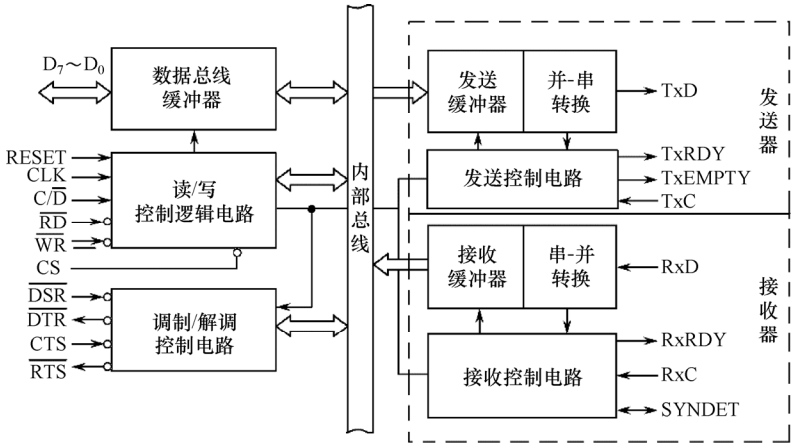


图 7-44　Intel 8251A 芯片内部结构

接收缓冲器：通过接收移位寄存器将到达 RxD 端的串行数据转成并行数据，同时由接收控制电路输出相应的状态。

发送缓冲器：完成“并-串”转换，从 TxD 端逐位输出。发送控制电路配合发送缓冲器工作，控制和管理所有与串行发送有关的功能。

收发速率：分别由 TxC、RxC 端上输入的时钟频率通过编程设定（1/16/64 分频）。

数据总线缓冲器：与系统数据总线相连，它传输发送和接收数据，也传输控制字、命令字和状态信息。

读/写控制逻辑电路：配合数据总线缓冲器工作。当 $\overline{WR}$ =0 时，将数据总线上的数据或控制字写入 8251A 芯片。当 $\overline{RD}$ =0 时，将数据或状态字读入 CPU。

C/$\overline{D}$ 信号：与 $\overline{WR}$ 、$\overline{RD}$ 信号配合确定数据、命令或状态。当 C/$\overline{D}$ =0 时，8251A 读/写的是数据；当 C/$\overline{D}$ =1 时，8251A 读/写的是状态字/控制字。通常用地址线（如 A_0）连接信号进行区分。

CLK 信号：提供 8251A 内部定时，并与 TxC、RxC 无关；RESET 信号使 8251A 强制进入复位状态。

调制解调控制电路：用于简化 8251A 和调制解调器的连接。

2．8251A 芯片的控制字及其工作方式

图 7-45 给出了 8251A 芯片的编程结构。从编程角度出发，8251A 芯片有数据输入和数据输出缓冲寄存器、发送和接收移位寄存器、1 个控制寄存器、1 个状态寄存器、1 个模式寄存器和 2 个同步字符寄存器。

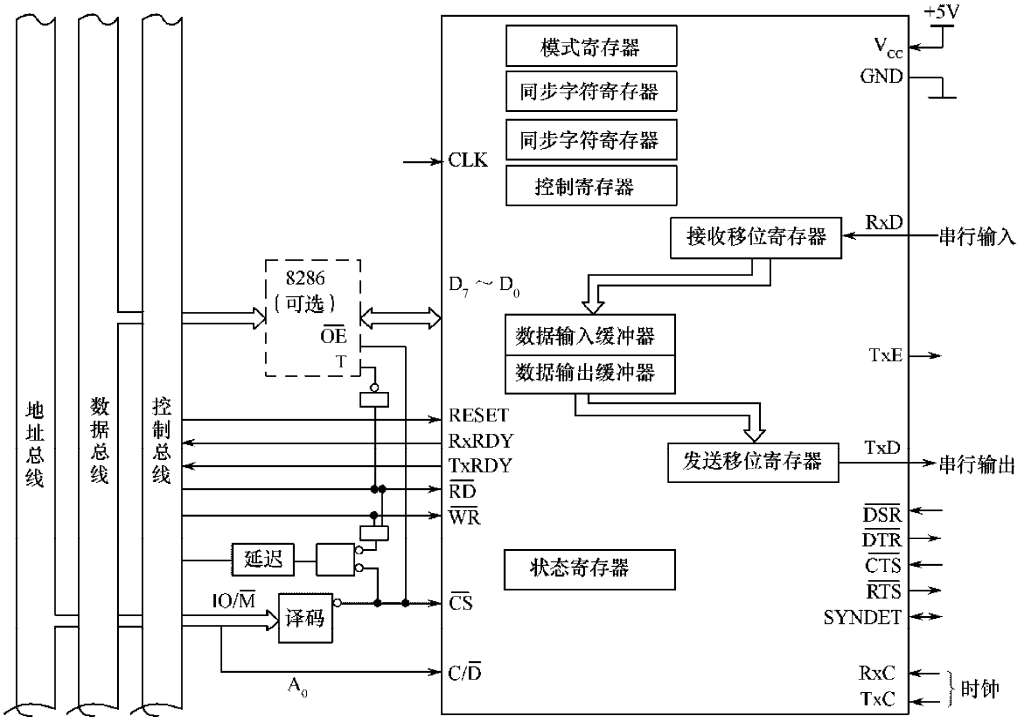


图 7-45　8251A 芯片的编程结构和外部连接

收发数据缓冲器合用一个端口地址，用输入/输出指令加以区分。当发送器空时，状态寄存器相应 TxRDY 位被置 1，TxRDY 引脚输出高电平。当接收缓冲器收到数据时，状态寄存器相应 RxRDY 位被置 1，RxDRY 引脚输出高电平。RxRDY 与 TxRDY 可作为中断信号向 CPU 申请中断服务。

状态寄存器由外设状态信息决定，同步字符寄存器用来存放同步方式中所用的同步字符。模式寄存器的内容决定了 8251A 芯片的传输方式是同步还是异步，并决定接收和发送字符的格式；控制寄存器用于控制 8251A 芯片的工作过程，这两个寄存器的内容均可通过程序修改设定。

8251A 芯片在使用前必须进行初始化，以确定它的工作方式、传送速率、字符格式以及停止位长度等，可使用的控制字如下。

（1）方式选择控制字

其格式如图 7-46 所示。B_2B_1 位定义 8251A 的同步/异步工作方式，如果是异步方式，还可由 B_2B_1 的取值来确定传送速率。×1 表示输入的时钟频率是波特率的 1 倍（相同），×16、×64 分别表示 16 倍、64 倍，因此通常称 1、16、64 为波特率系数，即

发送/接收时钟频率=发送/接收波特率×波特率系数

允许发送和接收波特率不同，RxC 和 TxC 也可不相同，但是它们的波特率系数必须相同。

L_2L_1：定义数据字符的长度，可为 5、6、7、8 位。

电子工业出版社

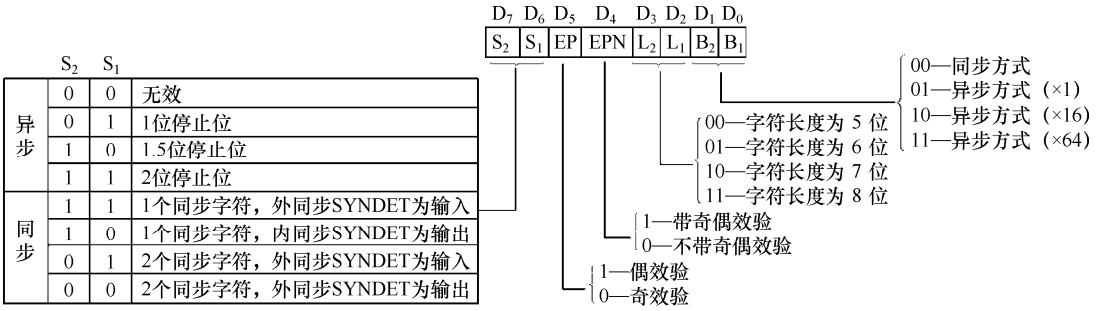


图 7-46　8251A 的方式选择控制字位结构

PEN：校验允许位，定义是否带奇/偶校验。PEN=1 时，由 EP 位定义是采用奇校验还是偶校验。

S_2S_1：定义异步方式的停止位长度（1、1.5、2 位）。对于同步方式，S_1=1 定义是外同步，S_1=0 定义是内同步。S_2=1 定义是单同步，S_2=0 定义是双同步。

（2）操作命令控制字。格式如图 7-47 所示。

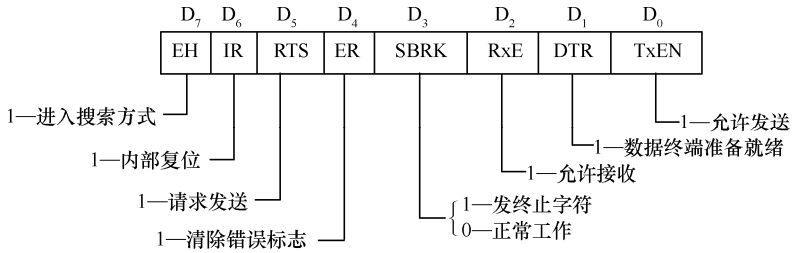


图 7-47　8251A 的操作命令控制字位结构

TxEN：允许发送位，TxEN=1 时，发送器才能通过 TxD 引脚向外部串行发送数据。

DTR=1：表示 CPU 已准备好接收数据，这时 $\overline{\text{DTR}}$ 引脚端输出有效。

RxE：允许接收位。RxE=1 时，接收器才能通过 RxD 引脚从外部串行接收数据。

SBRK：发送终止字符位。SBRK=1，通过 TxD 引脚连续发送 0 信号。正常通信过程中，SBRK=0。

ER：清除错误标志位。8251A 设置有三个出错标志：奇/偶校验标志 PE、溢出错误标志 OE 和帧校验错标志 FE。ER=1 时，将 PE、OE、FE 标志同时清 0。

RTS=1：迫使 8251A 输出 $\overline{\text{RTS}}$ 有效，表示 CPU 已做好发送数据准备，请求发送数据。

IR：内部复位信号。IR=1 时，迫使 8251A 回到接收方式选择控制字的状态。

EH：跟踪方式位，只对同步方式有效，EH=1，表示开始搜索同步字符，因此对于同步方式，一旦允许接收（RxE=1），必须同时使 EH=1，并且使 ER=1，清除全部错误标志，才能开始搜索同步字符。从此以后所有写入 8251A 的控制字都是操作命令控制字。只有外部复位命令 RESET=1 或内部复位命令 IR=1 才能使 8251A 回到接收方式选择命令字状态。

（3）状态控制字

CPU 可在 8251A 工作过程中读取当前 8251A 的状态字，其格式如图 7-48 所示。

PE：奇/偶错标志位。PE=1 表示当前产生了奇/偶错，它不中止 8251A 的工作。

OE：溢出错标志位。OE=1，表示溢出错，CPU 没有来得及将上一字符读走，又有新字符送到接收寄存器。它不中止继续接收下一字符，但上一字符将丢失。

電子工業出版社

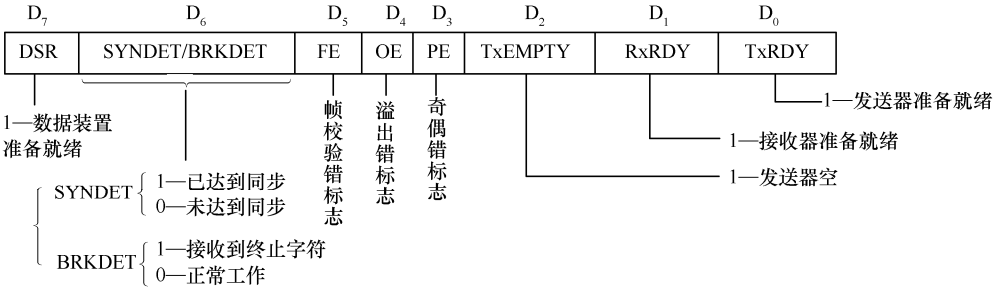


图 7-48　8251A 的状态字位结构

FE：帧校验错标志位。FE 只对异步方式有效。FE=1，表示未检测到停止位，不中止工作。

上述三个标志位允许用操作命令控制字中的 ER 复位。

TxRDY：与 TxRDY 引脚信号的含义有些区别。TxRDY=1，只反映当前发送数据缓冲器已空，而 TxRDY 引线端为 1，除发送数据缓冲器已空外，同时 $\overline{CTS}$ =0，TxEN=1。

在数据发送过程中，两者相同，通常 TxRDY 状态位供 CPU 查询，TxRDY 引线端可用作向 CPU 发出中断请求信号。

RxRDY、TxEMPTY、SYNDET/BRKDET：与同名引线端的状态完全相同，可供 CPU 查询。

DSR=1：表示外设或 Modem 已准备好发送数据，这时输入引线端 $\overline{DSR}$ 有效。

CPU 可在任意时刻读 8251A 状态字，这时 C/$\overline{D}$ 引线端输入应为 1，在 CPU 读状态期间，8251A 将自动禁止改变状态位。

对 8251A 芯片进行初始化编程，必须在系统复位之后，总是先写方式选择控制字，并且必须紧跟在复位命令之后。如果定义 8251A 工作于异步方式，必须紧跟操作命令控制字进行定义，然后才可开始传送数据。在数据传送过程中，可使用操作命令字重新定义，或使用状态控制字读取 8251A 的状态，待数据传送结束，必须用操作命令控制字将 IR 位置 1，向 8251A 传送内部复位命令后，8251A 才可重新接收方式选择命令字、改变工作方式，完成其他传送任务。

如果采用同步工作方式，在方式选择控制字之后就输出同步字符，在一个或两个同步字符之后再使用操作命令控制字，以后的过程同异步方式。

8251A 芯片初始化编程的操作过程用流程图描述，如图 7-49 所示。

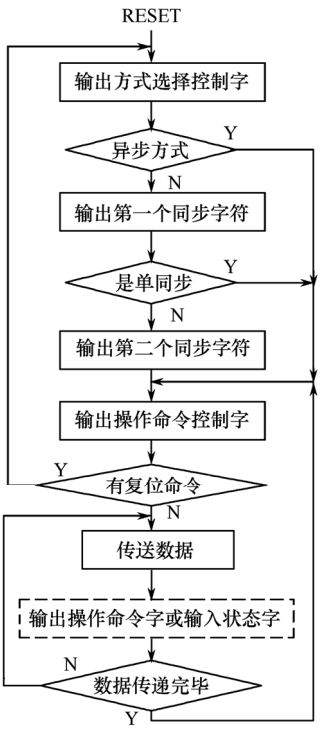


图 7-49　8251A 初始化流程图

3．8251A 芯片串行接口应用举例

（1）8251A 与 RS-232 接口

计算机或外设均可采用 8251A 实现 RS-232C 串行接口通信系统。每个 RS-232C 串行接口采用一片 8251A 芯片，其原理如图 7-50 所示，可采用异步或同步方式实现单工、双工或半双工通信。

当采用查询方式、异步传送实现双方半双工通信时，收/发两端 CPU 的初始化程序分别定义为接收器和发送器。发送器 CPU 查询到 TxRDY 有效，则向 8251A 并行输出 1 字符数

電子工業出版社

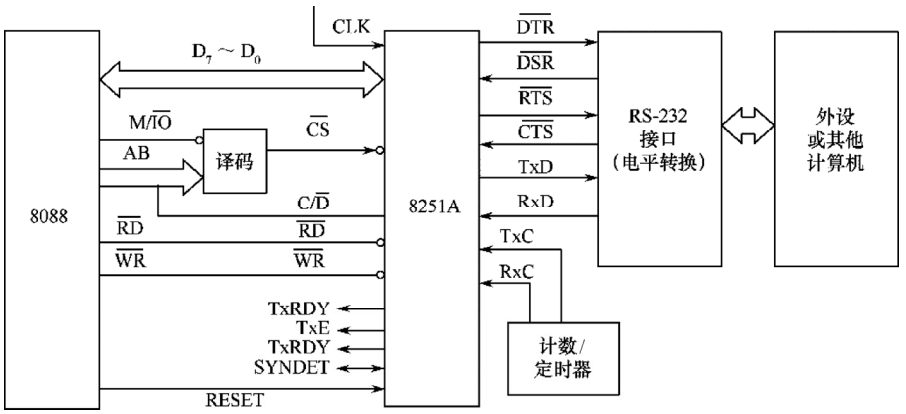


图 7-50　8251A 典型应用电路

据；接收端 CPU 查询到 RxRDY 有效，则从 8251A 并行输入 1 字符数据；一直进行到全部数据传送完毕为止。

发送端初始化程序与发送控制程序如下：

```
; CTRL8251 为 8251A 控制/状态端口，DATA8251 为 8251A 数据端口
; 接收数据块首地址 RBUF，发送数据块首地址 SBUF，发送数据块字节数/计数值 COUNT
SNDINI:  MOV     DX, CTRL8251
         MOV     AL, 7FH                  ; 将 8251A 定义为异步方式
         OUT     DX, AL                   ; 8 位数据，1 位停止位，偶校验
         MOV     AL, 11H                  ; 取波特率系数为 64，允许发送
         OUT     DX, AL
         MOV     SI, OFFSET SBUF          ; 发送数据块首地址，设置发送指针
         MOV     CX, COUNT                ; 发送数据块字节数/计数值
NEXT:    MOV     DX, CTRL8251
         IN      AL, DX                   ; 查询 TxRDY 有效否
         AND     AL, 01
         JZ      NEXT
         MOV     DX, DATA8251
         MOV     AL, [SI]                 ; 向 8251A 输出 1 字节数据
         OUT     DX, AL
         INC     SI
         LOOP    NEXT
         …
```

接收端初始化程序和接收控制程序如下：

```
RCVINI:  MOV     DX, CTRL8251
         MOV     AL, 7FH                  ; 初始化 8251A
         OUT     DX, AL
         MOV     AL, 04H
         OUT     DX, AL                   ; 置接收数据块指针和计数值
         MOV     DI, OFFSET RBUF
         MOV     CX, COUNT
RCVNEXT: MOV     DX, CTRL8251
         IN      AL, DX                   ; 查询 RxRDY 有效否
         ROR     AL, 1
```


電子工業出版社

```
        ROR     AL, 1
        JNC     RCVNEXT
        ROR     AL, 1                    ; 查询是否有奇偶校验错
        ROR     AL, 1
        JC      RCVERR
        MOV     DX, DATA8251
        IN      AL, DX                   ; 输入 1 字节到接收数据块
        MOV     [DI], AL
        INC     DI
        LOOP    RCVNEXT
        …
```

（2）串行打印机接口

以 TPμP-40S 微型打印机为例，其接口与 RS-232C 标准兼容，D 型 9 线插座的引脚信号定义如表 7-4 所示。波特率、字符格式和握手方式等可由机内 DIP 开关选择。

表 7-4　微型打印机 TPμP-40S 引脚信号

引脚	信号	源	说　明
1	DCD	打印机	同信号 CTS
3	TxD	主机	打印机从主计算机接收数据
2	RxD	打印机	当使用 X-ON/X-OFF 握手协议时，打印机向计算机发送控制码 X-ON/X-OFF
5	GND	—	信号地
6	DSR	打印机	该信号为“SPACE”状态时，表示打印机“在线”
8	CTS	打印机	该信号为“MARK”状态时，表示打印机正“忙”不能接收数据；该信号为“SPACE”状态时，表示打印机“准备就绪”可以接收数据

选微型打印机串行通信使用状态标志握手，数据格式为 8 位数据、无校验。微型打印机与 PC 主机的 8251A 接口之间单工传输，硬件连接如图 7-51 所示。查询式打印字符串“HELLO!”的子程序如下。

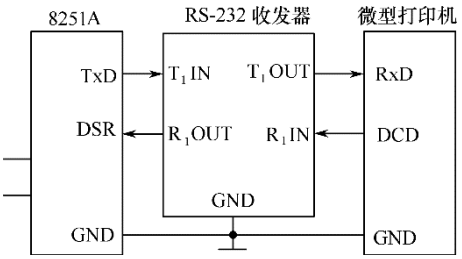


图 7-51　串行微型打印机与 8251A 接口

```
; 设异步通信的数据端口地址为 DATA51，控制状态端口 CTRL51
TAB     DB      'HELLO!', 0AH, 0DH
TAB_L   EQU     $-TAB
…
…
PRINT   PROC
        MOV     BX, OFFSET TAB
        MOV     CX, TAB_L
TRAN:   MOV     DX, CTRL51
WAIT1:  IN      AL, DX
```


電子工業出版社

```
        TEST    AL, 80H             ; 打印机忙（DSR=1）?
        JNZ     WAIT1
        MOV     DX, DATA51          ; 发送数据
        MOV     AL, [BX]
        OUT     DX, AL
        INC     BX
        LOOP    TRAN
        RET
PRINT   ENDP
```

7.4.3 可编程异步通信接口 INS8250

INS8250 芯片使用方便，具有以下基本功能：

① 完全双工、双缓冲器发送和接收器的异步通信接口电路。

② 通信波特率为 50～9600，共分成 15 档。

③ 异步通信格式可通过编程选择，即字符可选传送 5～8 位；可选 1、1.5、2 位停止位；并可产生中止字符（输出连续的低电平，以通知对方中止通信）。

④ 可选奇偶校验，并具有奇偶、溢出和帧错误等检测状态位。

⑤ 片内具有优先权中断控制逻辑。

INS8250 芯片具有 UART 的典型功能，内部结构与图 7-41 相近，配备通信收/发缓冲器、通信控制和状态寄存器、通信线路控制寄存器 LCR、通信线路状态寄存器 LSR、调制解调控制寄存器 MCR、调制解调状态寄存器 MSR、中断识别寄存器 IIR、中断允许寄存器 IER 等。INS8250 支持 4 级中断，但共用一条中断请求信号线 INTRPT，可通过中断识别寄存器 IIR 识别中断源。内置波特率发生控制及调制/解调控制电路，方便硬件设计和编程使用，因此它被早期 PC 选为标准串行接口配置，但目前多被通用串行总线 USB 所替代。

7.4.4 通用串行总线 USB

通用串行总线 USB（Universal Serial Bus）是一种新型的串行接口标准，自 1996 年诞生 USB 1.0 规范以后，又陆续公布了 USB 1.1、USB 2.0、USB OTG 和 USB 3.0 等规范，其中 USB 2.0 的传输带宽达到 480 Mbps，USB OTG 更使 USB 装置摆脱了原来主从架构的限制，实现了端对端的传输模式。USB 可以实现外设的简单、快速连接，达到方便用户、降低成本、扩展 PC 连接外设范围的目的，还可向低压设备提供 5 V 电源，减少微机 I/O 接口数量。目前 USB 已取代传统的 RS-232C 成为微机的标准串行接口，并已成为主要的移动外设接口。

1．主要特点

① 集线器（Hub）使用树形连接，外设安装十分方便，具有真正的“即插即用”（Plug and Play，PnP）特性，用户可以方便地进行外设的连接、安装和卸除，主机可按外设的增删情况自动配置系统资源，用户可在不关机情况下进行外设的更换，外设驱动程序的安装、删除实现了自动化。

USB 连接外围设备时可使用 Hub 进行树形连接，连接于 USB 上的装置不限定是终端点，也可是利用 Hub 连接其他装置的分叉点。此外，它连接的装置之间不是平等关系，上、下游

電子工業出版社

关系明确。

用 USB 连接的外部设备数目最多可达 127 个，节点间连接距离为 5 m，这对一般的计算机系统应用是足够的。

② 低成本。一方面，USB 外设的设计制造过程比较简单，属于一种开放的、没有专利版权限制的理想工业标准。另一方面，USB 从 1996 年 4 月起并入 Intel 芯片组，从而使设备制造成本降低。

③ 三类通信速率。USB 总线支持三种数据传输速率：高速信令位传输速率 480 Mb/s、全速率 12 Mb/s、低速率 1.5 Mb/s。USB 2.0 支持在主控制器与 Hub 之间用高速传输全速和低速数据，而 Hub 与设备之间以全速或低速传输数据，这种支持能力可以将全速和低速设备对高速设备可用带宽的影响减到最小；USB 3.0 进一步提高了 USB 总线的传输速率，并且支持 OTG 功能。

④ 三种传送模式对应多类装置。USB 有同步、中断、大批三种数据传送模式。同步传送主要用于数码相机、扫描仪等中速外围设备。中断传送供键盘、鼠标等低速装置使用。大批传送则供打印机、调制解调器等不定期传送大量数据的中速装置使用。

⑤ 不同时钟信号用 NRZI 使数据同步。采用 NRZI 编码方式使数据获得同步，其特点是容易实现同步，但由于没有分离的时钟信号，数据的可维护性不够。

⑥ 装置和 Hub 可从总线直接获得电源。

⑦ 作为 PC 标准接口，通过专用芯片实现各种串/并行连接适配功能，市售产品包括 RS-232、I^2C/SPI、CAN 总线、有线/无线以太网转换器等，大大方便了微机应用。

2．USB 系统的结构框架

一个经典 USB 系统可分为 4 部分：USB 主机、USB 设备、USB Hub 和 USB 互连。USB 系统中的设备与主机的连接方式采用星形连接，其拓扑结构如图 7-52 所示。

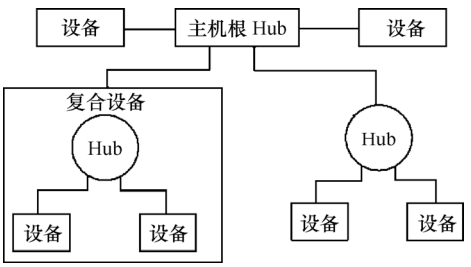


图 7-52　USB 系统的拓扑结构

Hub 是配线连接器，连接点被称为端口，每个 Hub 把单个连接点转换成多个连接点，这种结构支持多个 Hub 的连接。Hub 只有一个上游端口，其余为下游端口，通过上游端口连到主机，每个下游端口允许连接一个 Hub 或功能设备，直到所有外设加起来达到 127 个为止。多个不同功能的应用设备可作为复合设备放在一起，在主机看来，这个复合设备与一个带着若干设备的单独 Hub 是一样的。主机上有一组插口，主要作为根集线器（Root Hub）。根 Hub 是一个 USB 系统的总控制端口，既可以直接连接外设，也可通过 Hub 扩展控制更多的外设。

3．USB 物理接口

标准 USB 连接器分为两类：A 类和 B 类，其外形如图 7-53 所示。A 类连接器适合不经常插拔的设备，如键盘、鼠标、Hub 等；B 类连接器适合经常需要插拔的设备，如打印机、

電子工業出版社

扫描仪、调制解调器等。A 类和 B 类连接器不能互换。每类连接器都具有坚实的屏蔽外壳和易于插拔的特性。所有设备都有一个上行（Up-stream）连接点，Hub 则具有上行和下行（Down-stream）连接点，上游连接器和下游连接器在机械上不可互换，因此消除了 Hub 的非法环路连接。

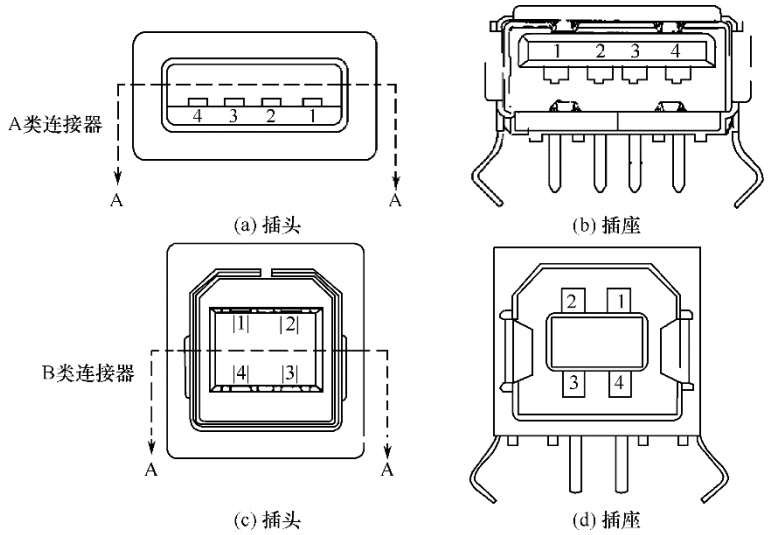


图 7-53　USB 总线 A/B 类连接器

USB 总线电缆有 4 条线：一对标准尺寸的双绞信号线和一对标准尺寸的电源线。USB 总线 A/B 类连接器引脚编号和信号名称如表 7-5 所示，每个引脚的最大电流不超过 1 A。D+和 D-是差分信号线，时钟随同差分数据一起编码、传送，时钟编码方案采用具有位填充的 NRZI 编码方式，每个通信包前面的同步字段允许接收器同步它们的位恢复时钟，总线特性阻抗是 90 欧姆。Vbus 和 GND 用来给设备提供电源，Vbus 在源端一般为+5 V。

表 7-5　标准 USB 总线连接器引脚编号

连接引脚	信号名称	注　释
1	Vbus	电源
2	D-	信号线-
3	D+	信号线+
4	GND	地

USB Type-C 规范简称 USB-C，支持 USB 接口双面插入，更纤薄（8.3 mm×2.5 mm）、更快的传输速度（最高 10 Gbps）、更强悍的电力传输（最高 100 W）。插座外观和信号如图 7-54 所示，包括 4 对电源/地，2 对高速 USB 数据，4 对屏蔽增强型超高速数据对，2 组边带应用 SBU（Sideband Use）引脚和 2 组配置通道 CC（Configuration Channel）引脚。

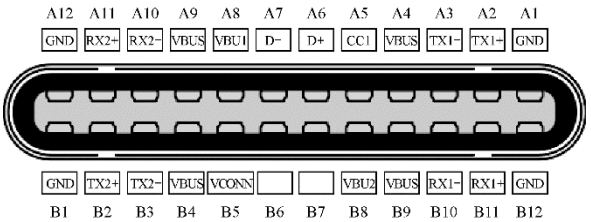


图 7-54　USB 总线 C 类连接器

4．USB 通信流与传输方式

USB 在客户软件及其 USB 设备之间提供通信服务，根据客户与 USB 设备之间不同的操

電子工業出版社

作情况，USB 设备可以有不同的通信流要求。通过向一个 USB 设备分配不同的通信流，USB 可以充分利用总线。每个通信流使用总线访问来完成客户与 USB 设备之间的通信。

USB 标准规定了 4 种传输方式：控制传输、等时序传输（Isochronous Transfer）、中断传输和数据块（Bulk）传输。每种传输类型确定通信流的多种特性，包括 USB 所用的数据格式、通信流的方向、包规限制、总线访问限制、延迟时间限制、请求的数据序列、错误处理等，以满足不同类型的传输需要。

USB 属于协议总线，通过软件实现各种功能，更多内容可参考 USB 规范（www.usb.org）。

5．USB 接口与设备开发

USB 控制器通常需要专门的接口芯片，各类新型微处理器都集成了相应的模块。Intel CPU 系列从 PII 开始即将 USB 控制器集成到芯片组中，可直接驱动应用，并成为 PC 的标准串行接口总线。与一般串行接口类似，CPU 与 USB 控制器进行通信时，可使用输入/输出寄存器。如控制器作为总线设备挂在 I/O 总线（如 PCI）上，还需编程配置相应的寄存器，即 USB 主控制器模块包含两组软件可以访问的寄存器组，即输入/输出寄存器组和可选用的 PCI（总线）配置寄存器。

主要的输入/输出寄存器有命令寄存器和状态寄存器。USB 命令寄存器是一个 16 位寄存器，给出 USB 主控器要执行的命令，对该寄存器进行一次写入操作将使 USB 主控器执行一次写入命令。USB 状态寄存器也是 16 位，反映 USB 主控制器在被读取时未处理的中断和所处的状态。

另外，USB 主控制器模块还带有中断请求寄存器（USB INTR）、反映当前帧编号的配置寄存器（FRNUM）、指定帧列表基地址的帧列表基地址配置寄存器（FLBASEADD）、用于启动起始帧（SOF）修正的配置寄存器、反映端口状态（挂起、线路状态、连接状态等）和进行端口控制（如端口复位、传输速率选择、恢复检测等）的配置寄存器（PORTSC）等。

6．USB OTG 接口

OTG（On-The-Go）技术 2001 年公布，用于在没有 Host 的情况下，实现不同设备或移动设备间的连接和数据交换，进而使 USB 应用从 PC 外设跨越到了消费电子产品和通信电子产品领域。根据 USB 接口所属协议，OTG 可分为 3 类。

- USB2.0 OTG：一种是 Micro 5PIN OTG，常见安卓手机 OTG 接口；另一种是 Mini 5PIN OTG，常见安卓手机和平板 OTG 接口。
- Micro USB3.0 OTG：部分 2016 年以前的安卓手机 OTG 接口。
- Type C OTG：部分在 2016 年以后的安卓手机 OTG 接口。

USB OTG 在完全兼容 USB 2.0 的基础上增加了电源管理（节省功耗）功能，允许设备既可作为主机，也可作为外设操作（两用 OTG）。OTG 两用设备完全符合 USB 2.0 标准，并提供一定的主机检测能力，支持主机通令协议（HNP）和对话请求协议（SRP）。在 OTG 中，初始主机设备称为 A 设备，外设称为 B 设备。可用电缆的连接方式来决定初始角色。

7.4.5 I^2C 与 SPI 串行总线

近年来，多种嵌入式处理器/微控制器（单片机）集成了 SPI 或 I^2C 接口，使之成为重要的串行总线，并有专门的转换芯片实现到 UART 或 USB 的转换。

電子工業出版社

1．I²C 总线

I²C（Inter-Integrated Circuit）总线是一种由 Philips 公司开发的两线式串行总线，用于连接微控制器及其外围设备。I²C 总线产生于 20 世纪 80 年代，最初为音频和视频设备开发服务器管理、设备组件状态通信。例如，管理员可对各个组件进行查询，以管理系统配置或掌握组件的功能状态，如电源和系统风扇；可随时监控内存、硬盘、网络、系统温度等多个参数，增加了系统的安全性，方便了管理。作为一个有专利的控制总线，I²C 已成为工业标准。

目前，很多半导体集成电路产品都集成了 I²C 接口。很多外围器件，如存储器（如 E²PROM XC24Cxx 系列）、ADC/DAC（如 TLC549、TLC5620、MAX187）、监控芯片（DS1302）等也提供 I²C 接口。I²C 总线最主要的优点是简单性和有效性。由于接口直接在组件之上，因此 I²C 总线占用的空间非常小，减少了电路板的空间和芯片引脚的数量，降低了互连成本。总线长度可达 7.5 m，并且能够以 10 kb/s 的最大传输速率支持 40 个组件。I²C 串行总线由两根信号线构成，即由数据线 SDA 和时钟线 SCL 实现数据的发送和接收。CPU 与被控 IC、IC 与 IC 之间双向传送，传输速率为 100 kb/s。每个电路和模块都有唯一的地址，在信息传输过程中，I²C 总线上并接的每一模块电路既是主控器（或被控器），又是发送器（或接收器），这主要取决于它所要完成的功能。CPU 发出的控制信号分为地址码和控制量两部分，地址码用来选址，即接通需要控制的电路，确定控制的种类。

在传送数据过程中，I²C 总线共有 3 种信号。

① 开始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。

② 结束信号：SCL 为低电平时，SDA 由低电平向高电平跳变，结束传送数据。

③ 应答信号：接收数据的 IC 在接收到 8 位数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后，等待受控单元发出一个应答信号；CPU 接收到应答信号后，根据实际情况做出是否继续传递信号的判断。

送到 SDA 线上的字节必须为 8 位长度，每次传送的字节数不受限制，每字节后必须跟一个应答位。数据传送时，先传送最高位，如果接收设备不能接收下一字节，如正在处理一个外部中断，可以把 SCL 线拉成低电平，迫使主机处于等待状态。当从机准备好接收下一字节时再释放时钟线 SCL，使数据传送继续进行。

2．SPI 总线

串行外围设备接口 SPI（Serial Peripheral Interface）总线是 Motorola 公司推出的一种同步串行接口。许多微控制器（如 MC68xx）、DSP 及 DSP 控制器（如 TI 公司的 TMS 系列）、ARM 架构处理器等也都配有 SPI 硬件接口，可以方便地实现一个主器件与一个或多个从器件相连，一些采用串行接口的存储器和转换芯片（如 Microchip 公司的 25Cxxx E²PROM、Maxim 公司的 DAC MAX538、ADC MAX187 等）也使用该总线，部分应用参见 7.6 节。

SPI 使用两根数据线，一根串行时钟线和一个片选信号线。

- 串行数据输入：主入从出，简称 MISO。
- 串行数据输出：主出从入，简称 MOSI。
- 串行时钟：记为 SCLK 或 SCK、SCL。
- 从器件使能信号：记为 $\overline{SS}$ 或 SS#、NSS。 SPI 协议是一种主-从设置，SPI 主控把 $\overline{SS}$ 拉低以开始传输，然后驱动串行时钟 SCLK 把数据同步输入/输出到从设备。SPI 主控通过把 $\overline{SS}$ 拉回到高电平，以终止传输。

電子工業出版社

SPI 有两种工作模式：主模式和从模式，无论哪种模式，都支持 3Mb/s 的传输速率，并且提供为传输完成标志和写冲突保护标志置位的功能。在从模式下，可使用简化的三线 SPI，即由$\overline{SS}$、SCLK、DATA 构成，与 I^2C 类似，但多了$\overline{SS}$。SPI 接口有 4 种时钟模式，反映了 2 个信号 CPOL（时钟极性）和 CPHA（时钟相位）的 2 个状态，以 CPOL、CPHA 的形式表示。

7.5 DMA 控制器接口

DMA（Direct Memory Access）是指外部设备直接与计算机存储器进行数据交换的 I/O 方式。这种方式下，数据的 I/O 操作不需要 CPU 执行指令，也不经过 CPU 的内部寄存器，而是由 DMA 控制器（DMAC）控制，利用系统的数据总线在外设和存储器之间进行顺序读出、写入操作。这种方式可以达到很高的传输速率，因而广泛用于高速 I/O 设备接口。

正常情况下，系统总线归 CPU 控制，DMA 控制器作为普通接口芯片接受 CPU 的管理，在需要 DMA 操作时向 CPU 提出 DMA 请求（DREQ）；在 CPU 响应请求后，DMAC 可从 CPU 处获得对总线的控制权。DMAC 根据数据传输的起始地址和长度，提供对存储器或 I/O 设备的读/写命令，每传输一个数据后自动修改地址和待传数据的长度。数据传输完毕，发出 DMA 传送过程结束信号 EOP，也可由外界输入 EOP 信号中止 DMA 过程。

在功能上，一方面可编程 DMAC 能以主控方式接管 CPU 总线，直接在 I/O 设备与存储器之间进行连续、顺序的数据读/写操作，因此它有别于一般的 I/O 控制器；另一方面，作为一个可编程 I/O 器件，在未获得总线控制权时，可由 CPU 对它进行初始化编程，按通道设置数据传送类型和方式、内存首地址、自动地址增/减选择、数据长度等参数，CPU 执行 I/O 指令对 DMAC 相关寄存器写入和读出时，DMAC 与其他 I/O 电路一样，是连接在 CPU 总线上的从属器件。本节主要介绍 PC 中常用的可编程 DMA 控制器 Intel 8237A。

7.5.1 8237A 芯片的基本功能和引脚特性

Intel 8237A 芯片具有 4 个可编程独立通道，每个通道支持 64 KB 的数据块传输，5 MHz 工作频率下传输速率可达 1.6 MB/s，是典型的可编程 DMA 控制器。8237A 芯片的内部结构如图 7-55 所示。

1．8237A 芯片的基本功能

① 4 个独立的 DMA 通道。每个通道有 16 位基地址寄存器、16 位字节计数寄存器、16 位当前地址寄存器、16 位当前字节计数寄存器、6 位方式寄存器各 1 个。方式寄存器接收并保存来自 CPU 的方式控制字，使本通道工作于指定的方式。

② 定时和控制逻辑电路。在 DMA 请求服务之前，对 CPU 编程给定的命令字和方式控制字进行译码，以确定 DMA 的工作方式，并产生所需的定时信号。

③ 优先权编码和旋转优先权逻辑。对通道进行优先权编码，在同时接收不同通道的 DMA 请求时，确定响应的先后次序。通道的优先权可以通过编程确定是固定的或旋转的。

④ 共用寄存器。除了每个通道的寄存器，整个芯片还有一组公用的寄存器：1 个 16

電子工業出版社

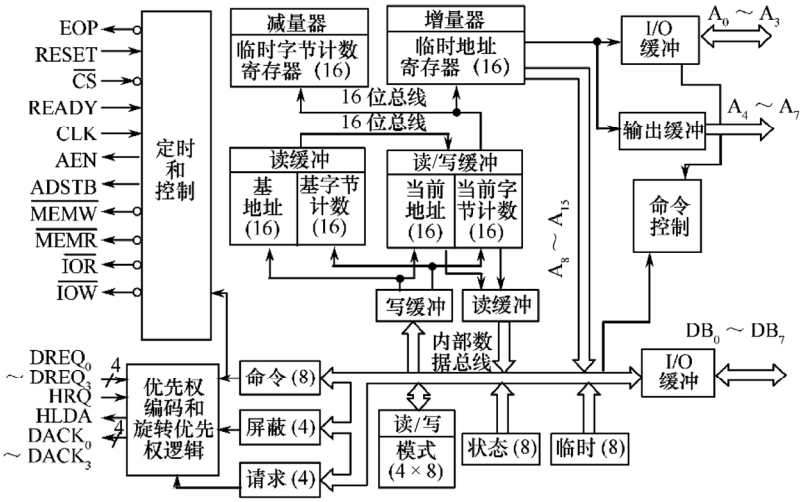


图 7-55　8237A 芯片的内部结构

位临时地址寄存器、1 个 16 位临时字节计数寄存器、1 个 8 位状态寄存器、1 个 8 位命令寄存器、1 个 8 位临时寄存器、1 个 4 位屏蔽寄存器、1 个 4 位请求寄存器等。8237A 内部寄存器的类型和数量如表 7-6 所示，其中地址、字节数临时寄存器等数量为 1 的寄存器为各通道所公用，其余数量为 4 的寄存器每个通道各有 1 组。

⑤ 8237A 数据总线。数据总线带有三态缓冲器，供读/写使用。

表 7-6　8237A 芯片内部寄存器

寄存器名	长度（b）	数量	寄存器名	长度（b）	数量
基地址寄存器	16	4	状态寄存器	8	1
基字节计数寄存器	16	4	命令寄存器	8	1
当前地址寄存器	16	4	临时寄存器	8	1
当前字节计数寄存器	16	4	方式寄存器	6	4
临时地址寄存器	16	1	屏蔽寄存器	4	1
临时字节计数寄存器	16	1	请求寄存器	4	1

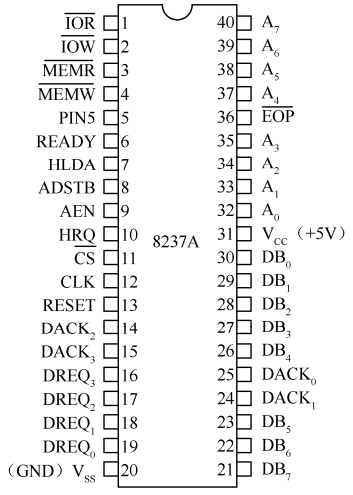


图 7-56　8237A 芯片的引脚

2．8237A 芯片的引脚结构

如图 7-56 所示，8237A 主要引脚定义如下。

CLK：时钟信号输入。对于标准的 8237A，输入时钟频率为 3 MHz；对于 8237A-2，输入时钟频率可达 5 MHz。

$\overline{CS}$：芯片选择信号，低电平有效。

RESET：复位信号，用来清除 8237A 中的命令、状态请求和临时寄存器，且使字节指针触发器复位并置位屏蔽触发器的所有位（即使所有通道工作在屏蔽状态）。在复位之后，8237A 工作于空闲周期。

READY：外设向 8237A 提供的高电平有效的“准备就绪”信号。若 8237A 在 S_3 状态以后的时钟下降沿检测到 READY 为低电平，则说明外设还未准备好下一次 DMA 操作，需要插入 S_W 状态，直到 READY 引脚出

现高电平为止。

$DREQ_0$～$DREQ_3$：DMA 请求信号。对应于 4 个独立通道，DREQ 的有效电平可以通过编程来确定，优先级可以固定或旋转。

$DACK_0$～$DACK_3$：对相应通道 DREQ 请求的应答信号。

DB_0～DB_7：8 条双向三态数据总线。在 CPU 控制系统总线时，可以通过 DB_0～DB_7 对 8237A 编程或读出 8237A 的内部状态寄存器的内容；在 DMA 操作期间，由 DB_0～DB_7 输出高 8 位地址信号 A_8～A_{15}，并利用 ADSTB 信号锁存该地址信号。在进行内存不同区域之间的 DMA 传送时，除了送出 A_8～A_{15} 地址信号外，还分时输入从存储器源区域读出的数据，送入 8237A 的临时寄存器中，等到存储器写周期时，再将这些数据通过这 8 个引脚，由 8237A 的临时寄存器送到系统数据总线上，然后写入规定的存储单元中。

A_3～A_0：低 4 位地址信号，双向、三态。在空闲周期，接收来自 CPU 的 4 位地址信号，用以寻址 8237A 内部的不同寄存器（组）；在 DMA 传送时，输出要访问的存储单元或 I/O 端口地址的低 4 位。

A_7～A_4：三态地址信号。在 DMA 传送时，输出要访问的存储单元或 I/O 端口地址的高 4 位。

$\overline{IOR}$：双向、三态 I/O 读信号，低电平有效。在空闲周期，它是一条输入控制信号，CPU 利用它读取 8237A 内部状态寄存器的内容；而在 DMA 传送时，它是读端口控制信号输出引脚，与 $\overline{MEMW}$ 配合，使数据由外设传送到内存。

$\overline{IOW}$：双向、三态 I/O 写信号，低电平有效，其功能与 $\overline{IOR}$ 相对应。

$\overline{MEMR}$：双向、三态存储器读信号，低电平有效，用于 DMA 传送。

$\overline{MEMW}$：双向、三态存储器写信号，低电平有效，用于 DMA 传送。

AEN：地址使能信号，高电平有效，由它把锁存在外部锁存器中的高 8 位地址送入系统的地址总线，同时禁止其他系统驱动器使用系统总线。

ADSTB：地址选通信号，高电平有效，把 DB_7～DB_0 上输出的高 8 位地址信号锁存到外部锁存器中。

$\overline{EOP}$：过程结束信号，双向。当字节数计数器减为 0 时，在引脚上输出一个有效的低电平脉冲，表明 DMA 传送过程已经结束；也可接收外部的 $\overline{EOP}$ 信号，强行结束 8237A 的 DMA 操作或重新进行 8237A 的初始化。当不使用 $\overline{EOP}$ 端时，应通过上拉电阻接到高电平上，以避免干扰。

其他还有 Vcc（+5 V）、Vss（GND）及 N/C 引脚（引脚 5，建议上拉）等。

7.5.2 8237A 芯片内部寄存器与编程

8237 芯片有 4 个独立的 DMA 通道，多个内部寄存器如表 7-6 所示，其功能和作用如下。

① 基地址寄存器：存放 16 位地址，只可写入，不能读出。在编程时，与当前地址寄存器被同时写入起始地址，可用作内存区域的首地址或末地址。在 8237A 进行 DMA 数据传送过程中，其内容不发生变化，只是在自动预置时，其内容可被重新写到当前地址寄存器中。

② 基字节计数寄存器：存放相应通道需要传送数据的字节数，只可写入，不能读出。在编程时它与当前字节计数寄存器被同时写入要传送数据的字节数。在 8237A 进行 DMA 数据传送过程中，其内容保持不变，只是在自动预置时，其内容可以被重新写到当前字节计数

電子工業出版社

寄存器中。

③ 当前地址寄存器：存放 DMA 传送期间的地址值。每次传送后自动加 1 或减 1。CPU 可以对其进行读/写操作。在选择自动预置时，每当字节计数值减为 0 或外部 $\overline{EOP}$ 有效时，就自动将基地址寄存器的内容写入当前地址寄存器，恢复其初始值。

④ 当前字节计数寄存器：存放当前字节数。每传送 1 字节，该寄存器的内容减 1。当设为自动预置时，若计数值减为 0 或接收到来自外部的 $\overline{EOP}$ 信号，会自动将基字节计数寄存器的内容写入该寄存器，作为初始计数值。

⑤ 临时地址寄存器和临时字节计数寄存器：中间过程使用，不需对其进行读/写操作。

⑥ 方式寄存器（地址偏移+0BH）：每个通道有 1 个 8 位方式寄存器，但是它们占用同一个端口地址，用来存放方式控制字，依靠方式控制字本身的特征位来区分写入不同的通道，规定通道的工作方式，各位的作用如图 7-57 所示。

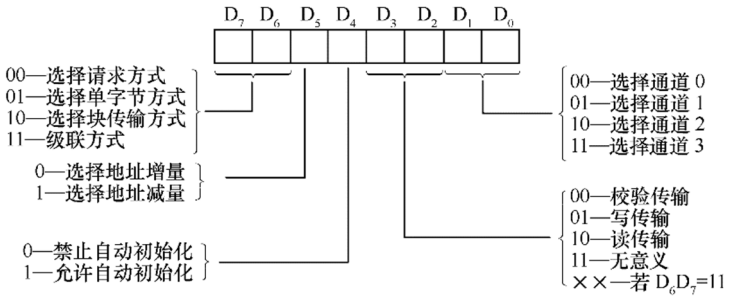


图 7-57　8237A 方式寄存器

自动预置就是当某一通道按要求将数据传送完后，又能自动预置初始地址和传送的字节数，以便重复前面已进行的过程。

校验传送指实际并不进行传送，只产生地址并响应 $\overline{EOP}$ 信号，不产生读/写控制信号，用以校验 8237A 的功能是否正常。

⑦ 命令寄存器（+08H，写）。存放编程的命令字，命令字如图 7-58 所示，位定义如下。

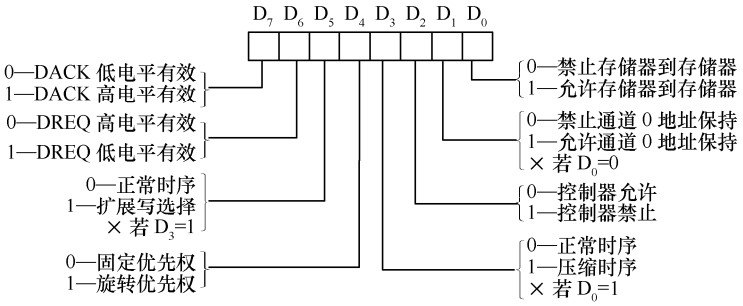


图 7-58　8237A 命令寄存器

D_0：规定是否允许采用存储器到存储器的传送方式。若允许，则利用通道 0 和通道 1 来实现。

D_1：规定通道 0 的地址是否保持不变。如前所述，在存储器到存储器传送中，源地址由通道 0 提供，读出数据到临时寄存器，而后由通道 1 送出目标地址，将数据写入目标区域；若命令字中 D_1=1，则在整个数据块传送中（块长由通道 1 决定）保持内存源区域地址不变，因此就会把同一个数据写入到整个目标存储器区域中。

D_2：允许或禁止 8237A 芯片工作控制位。

電子工業出版社

D_3：选择总线周期中写信号的定时。在 PC 中，动态存储器的写操作是由写信号的上升沿启动的，若在 DMA 周期中写信号来得太早，可能造成错误，所以 PC 选择 D_3=0。命令字的其他位容易理解，不再说明。

D_5：选择是否扩展写信号。在 D_3=0（正常时序）时，如果外设速度较慢，有些外设用 8237A 送出的 $\overline{\text{MEMW}}$ 和 $\overline{\text{IOW}}$ 信号的下降沿来产生 READY 信号。为提高传送速度，使 READY 信号早些到来，须将 $\overline{\text{MEMW}}$ 和 $\overline{\text{IOW}}$ 信号加宽，以使它们提前到来。因此可以通过令 D_5 =1 使 $\overline{\text{IOW}}$ 和 $\overline{\text{MEMW}}$ 信号扩展 2 个时钟周期，以便提前到来。

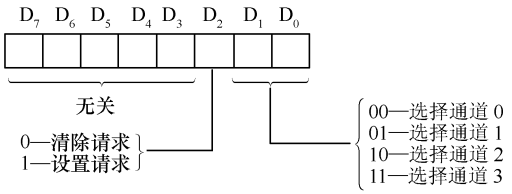


图 7-59　软件请求字格式

⑧ 请求寄存器（+09H）：在软件控制下产生一个 DMA 请求，等效外部 DREQ 请求。请求字格式如图 7-59 所示，D_0D_1 的不同编码表示向不同通道发出 DMA 请求。在软件编程时，这些请求是不可屏蔽的，利用命令字可使 8237A 按照命令字的 D_0D_1 所指的通道，完成 D_2 所规定的操作，这种软件请求只用于通道工作在数据块传送方式下。

⑨ 屏蔽寄存器。8237A 的屏蔽字有两种形式。

- 单个通道屏蔽字（+0AH）：格式如图 7-60 所示。利用这个屏蔽字，每次只能选择一个通道。其中 D_0D_1 的不同编码指示所选的通道，D_2=l 表示禁止该通道接收 DREQ 请求，D_2=0 表示允许 DREQ 请求。
- 4 通道屏蔽字（+0FH）：可以利用这个屏蔽字同时对 8237A 的 4 个通道的屏蔽字进行操作，故又称为主屏蔽字。格式如图 7-61 所示。它与单通道屏蔽字占用不同的 I/O 接口地址。

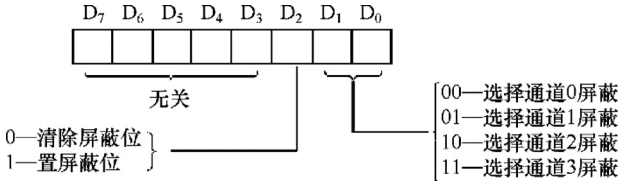


图 7-60　8237A 单通道屏蔽寄存器

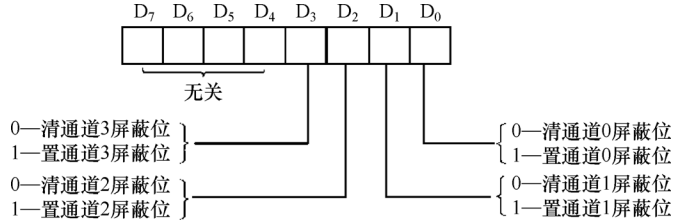


图 7-61　8237A 4 通道屏蔽寄存器

⑩ 状态寄存器（+08H，读）：存放各通道的状态，CPU 读出其内容后，可得知 8237A 的工作状态。包括：通道传输结束，计数已达到计数终点，对应位为 1；通道的 DMA 请求尚未处理，对应位为 1。状态寄存器格式如图 7-62 所示。

⑪ 临时寄存器（+0DH，读）：用于存储器到存储器传送过程中数据的暂时存放。

電子工業出版社

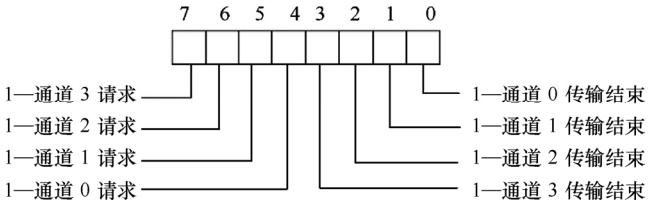


图 7-62　8237A 状态寄存器

⑫ 字节指针触发器（+0CH）：一个特殊触发器，用于对前述各 16 位寄存器寻址。由于寄存器的读/写必须分两次进行，先低字节，后高字节，为此要利用该触发器，当此触发器状态为 0 时，进行低字节操作。低字节读/写操作完成后，触发器自动置 1，再次操作又清 0。利用这种机制实现双字节读/写操作，16 位寄存器仅占用一个外设端口地址。

⑬ 主清除（复位）寄存器（+0DH，写）：与硬件复位等效，进入空闲周期。

7.5.3　8237A 应用与编程

DMA 的典型应用包括：利用存储器通过数模转换器（DAC）产生连续波形、由 ADC 高速转换存储到存储器中实现高速连续采样、高速通信中网卡与存储器高速数据交换等。8237A 应用在早期的 PC/XT 中，用于软盘、硬盘快速交换和用户扩展（I/O 总线、网卡等）。通道 0 用于定时动态刷新 DRAM，刷新周期 T=15.08 μs，优先级最高，这时仅允许单字节方式。

1．8237A 芯片的初始化

PC 系统的 BIOS 对 8237A 芯片进行初始化和测试，测试程序对 8237A 的 4 个通道的 8 个 16 位寄存器先后分别写入 0FFFFH 和 0000H，再读出进行比较，以检测 8237A 是否工作正常。BIOS 测试程序段（DMA 端口地址=00H）如下：

```
        MOV     AL，04              ；命令字禁止 8237A 工作
        OUT     DMA+08H，AL         ；命令字送命令寄存器
        OUT     DMA+0DH，AL         ；写总清命令，8237A 进入空闲周期
        ；
        MOV     AL，0FFH            ；0FFH→AL，8 位作全 1 检测
T1:     MOV     BL，AL              ；保存到 BX 寄存器
        MOV     BH，AL
        MOV     CX，08              ；检测次数
        MOV     DX，DMA             ；0FFH 写入 0#～3#通道的地址或字节计数寄存器
T2:     OUT     DX，AL              ；写低 8 位
        OUT     DX，AL              ；写高 8 位
        IN      AL，DX              ；读出写入的低 8 位
        MOV     AH，AL              ；保存到 AH
        IN      AL，DX              ；读出高 8 位
        CMP     BX，AX              ；读出内容是否与写入内容相同
        JZ      T3                  ；相同，转 T3 继续
        JMP     ERROR               ；不同，8237A 有错停机
T3:     INC     DX                  ；寄存器地址加 1，检查下一寄存器
        LOOP    T2
        INC     AL                  ；使(AL)=0
```


電子工業出版社

```
        JZ      T1                      ; 返回作全 0 检测
        …
```

在 PC 中，DRAM 的刷新通过 8237A 的 0#通道来实现，其定时则通过 8253 芯片的 1#通道来提供，下面是 PC BIOS 对 DRAM 刷新的初始设置程序（TIMERI=40H）：

```
        …
        MOV     AL, 0FFH                ; 0#通道传送字节数
        MOV     DX, DMA+1
        OUT     DX, AL                  ; 写基值和现行字节计数寄存器低 8 位
        OUT     DX, AL                  ; 写高 8 位
        MOV     AL, 58H                 ; DMA 工作方式字
        OUT     DMA+0BH, AL             ; 写入方式寄存器
        MOV     AL, 0                   ; DMA 命令字
        MOV     CH, AL                  ; 计数次数高 8 位为 0
        OUT     DMA+8, AL               ; 写命令字到命令寄存器
        OUT     DMA+10, AL              ; 写屏蔽寄存器，允许 0#通道请求
        MOV     AL, 54H                 ; 系统 8253 定时/计数器控制字
        OUT     TIMERI+3, AL            ; 写入 8253 控制字
        MOV     AL, 12H                 ; 设置 8253 的 l#通道计数初值
        OUT     TIMERI+1, AL            ; 写入 1#通道计数初值
        …
```

2．8237A 的应用

【例 7-4】把存储器从 80000H 开始的 300H 字节通过 8237A 通道 1 送往 I/O 端口。

```
        XOR     AL, AL                  ; DMAC 地址 00H～0FH
        OUT     0DH, AL                 ; 主复位寄存器 MR 复位
        OUT     02, AL                  ; A0～A7 当前地址→CH1 寄存器
        OUT     02, AL                  ; A8～A15
        MOV     AL, 8                   ; A16～A19
        OUT     83H, AL                 ; 存储器页面寄存器，这是微机的规定
        MOV     AX, 300H-1              ; 计数值
        OUT     03H, AL
        MOV     AL, AH
        OUT     03H, AL
        MOV     AL, 49H                 ; 单字节读，+1 计数
        OUT     0BH, AL
        MOV     AL, 40H                 ; 命令字
        OUT     08H, AL                 ; 通道 1，低电平有效，正常时序
        MOV     AL, 01                  ; 允许 DMA#1
        OUT     0AH, AL                 ; 写屏蔽字
WAITF:  IN      AL, 08H
        TEST    AL, 02
        JZ      WAITF
        MOV     AL, 05
        OUT     0AH, AL                 ; 结束，屏蔽 DMA#1
```


電子工業出版社

7.6 模拟量输入/输出接口

7.6.1 概述

自然环境中，除了离散的数字量，计算机与外界交互更多地通过连续变化的模拟量信号进行。在一个实际的计算机测控系统（如图 7-63 所示）中，计算机作为系统的核心环节，输入/输出都是数字信号，而外部受控对象往往是连续变化的模拟系统，传送的主要是连续的模拟信号。模拟量输入/输出通道是计算机与应用对象之间的一类重要接口，也是计算机测控与实时处理系统重要的组成部分。

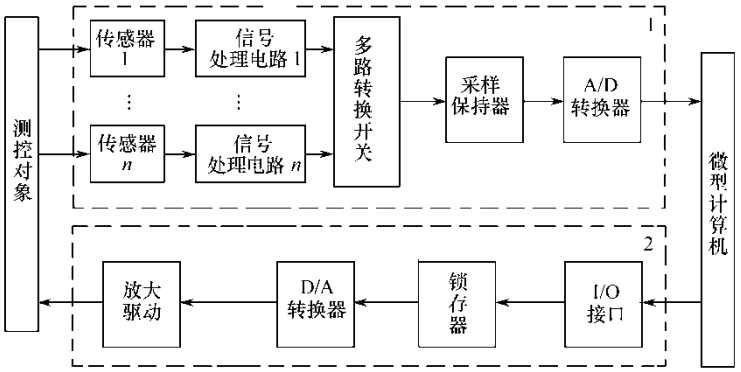


图 7-63　计算机控制系统与模拟量输入/输出通道

典型的模拟量输入/输出通道如图 7-63 中虚线框部分所示，虚线框 1 对应模拟量输入通道，虚线框 2 为模拟量输出通道。接口的核心是数/模（D/A，Digit to Analog）和模/数（A/D，Analog to Digit）转换器，实现两类信号变换。系统中完成模拟信号转换成数字信号的装置称为 A/D 转换器（ADC）；反之，完成数字信号转换成模拟信号的装置称为 D/A 转换器（DAC）。A/D 转换器接口也称为模入通道接口或 A/D 接口，D/A 转换器接口也称为 D/A 接口或模出通道接口。两部分合称模拟通道接口。

目前，A/D 和 D/A 转换多用独立芯片来完成，也可与 CPU 集成在一个芯片中，如很多嵌入式处理器、数字信号处理器（DSP）或微控制器（MCU，单片机）内置有模拟接口。这种高集成度的芯片可使其可靠性提高，成本下降，应用则更加简单，但转换精度等性能一般。

典型的模拟量输入通道包括传感器、信号调理/处理环节（电路）、多路转换开关、采样保持器、模数转换器等。

传感器把非电物理量转换成电信号（电流或电压），通过信号放大调理环节达到常用或标准的电压等级，如双极性的 0～±5 V、0～±10 V，单极性的 200 mV、0～2.00 V、0～+5 V、0～+10 V 等，供 A/D 转换用。放大调理电路由各种运算放大器和基准电压源（零位迁移用）构成，如 ICL7650（自稳零斩波放大）、OP07（低失调精密运放）、BB175（仪表放大器）等。

多路转换开关则是考虑到当多路信号需要采集时，多路复用、分时采样可降低成本和简化结构。可用机械或电子开关，一般多个模拟信号共用一个模数转换器（n 选 1 开关），开关可以是单向或双向的。常见专用芯片有 CD4051B、LF13508、AD7501/3（均为 8 选 1）等。

采样保持器是为了保证在 A/D 转换过程中保持待转换的输入信号不变的专门电路。常用的单片采样保持器有 LF398、AD582K 等。

为了保证转换接口电路的精度，模拟接口通道中需要专门的高精度电压基准源（Vref,

電子工業出版社

Voltage reference），有时还需要考虑适当的电气隔离措施。

在输出方面，由于计算机直接输出的信号以数字形式给出，而有的执行部件要求提供连续的电流或电压信号，故需要采用模拟量输出通道实现，这个任务需要由数模转换器完成。由于输出转换的信号需要保持，可以与模入通道类似，使用采样保持器和多路转换开关（通过电容保持信号，并需要反复刷新），但更多的是通过锁存器来保持数字量不变。

输入/输出信号一般还需要低通滤波器，滤除干扰信号或保持输出平滑。为了实现远距离传输，可能需要进行电压/电流（V/I）变换，或进行功率放大，以便驱动外部设备。

7.6.2 并行和串行 D/A 转换器

1．D/A 转换器工作原理

D/A 转换器的电路形式较多，最常见的是 T 形电阻网络。图 7-64 是一个 4 位 D/A 转换器，数字量的 D_3～D_0 分别控制一个模拟开关，当某一位为 1 时，对应开关倒向右边（加权）；反之，开关倒向左边（接地）。考虑运算放大器工作在负反馈方式，容易分析出图中 X_3～X_0 各点对应的电位分别为 V_{ref}、$V_{ref}/2$、$V_{ref}/4$、$V_{ref}/8$，而与开关方向无关。于是有

$$\sum I = \frac{V_{X_3}}{2R}\cdot D_3 + \frac{V_{X_2}}{2R}\cdot D_2 + \frac{V_{X_1}}{2R}\cdot D_1 + \frac{V_{X_0}}{2R}\cdot D_0 \qquad (D_3, D_2, D_1, D_0 = 0\text{或}1)$$

$$= \frac{1}{2R\times 2^3} V_{ref}(D_3\times 2^3 + D_2\times 2^2 + D_1\times 2^1 + D_0\times 2^0)$$

$$V_o = -R_f\cdot\sum I = -\frac{R_f}{2R\times 2^3}\cdot V_{ref}\cdot\sum_{i=0}^{3} D_i\times 2^i$$

也就是说，输出电压正比于数字量的值，并需要基准电压 V_{ref} 直接保证精度。

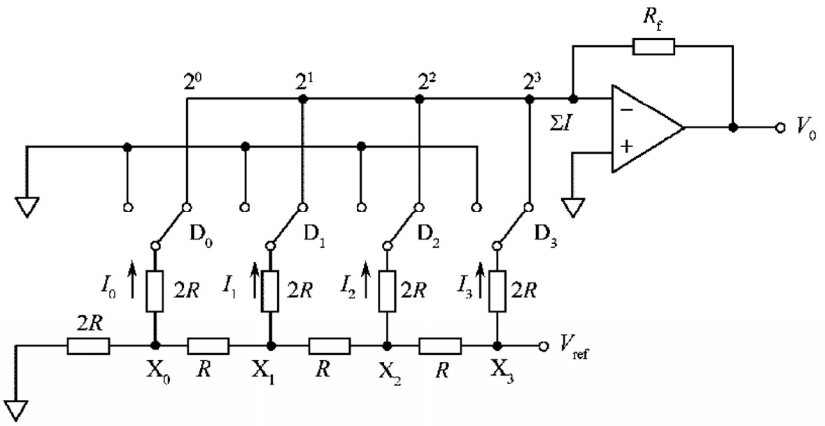


图 7-64　T 形电阻网络构成的 4 位 DAC

2．D/A 转换器的主要性能指标

① 分辨率：指 D/A 转换器所能分辨最小量化信号的能力，是对微小输入量变化的敏感程度的描述，一般用转换器数字量的位数来表示。对于一个分辨率为 n 位的 DAC，它能对满刻度的 2^{-n} 倍的输入变换量做出反应。常见的分辨率有 8、10、12、14、16 位等。

② 建立时间：DAC 转换速度快慢的重要参数，指 DAC 的数字输入发生满刻度值的变化时，其输出模拟信号电压（或电流）达到满刻度值 ± 1/2 LSB 时所需要的时间。对采用电流输出形式的 DAC，其建立时间是很短的；而对电压输出形式的 DAC，其建立时间主要取决

電子工業出版社

输出运放所需的响应时间。一般 DAC 的建立时间为几 ns 至几μs。

其他还有绝对精度、相对精度、线性度、温度系数和非线性误差等性能指标。

3．典型 8 位 DAC 芯片——DAC0832

DAC0832 是美国 NSC 公司生产的 8 位双缓冲 D/A 转换器，片内带有数据锁存器，可与通常的微处理器直接接口。电路有良好的温度跟随性，并使用 CMOS 电流开关和控制逻辑来获得低功耗和低输出泄漏电流误差。

（1）主要性能指标

8 位分辨率；建立时间 1 μs；电流型输出；单电源供电+5 V～+15 V；低功耗 200 mW；基准电压范围±25 V。

（2）内部结构和引脚

DAC0832 由 8 位输入锁存器、8 位 DAC 寄存器、8 位 D/A 转换电路组成，内部逻辑结构如图 7-65 所示，引脚信号及含义如图 7-66 和表 7-7 所示。

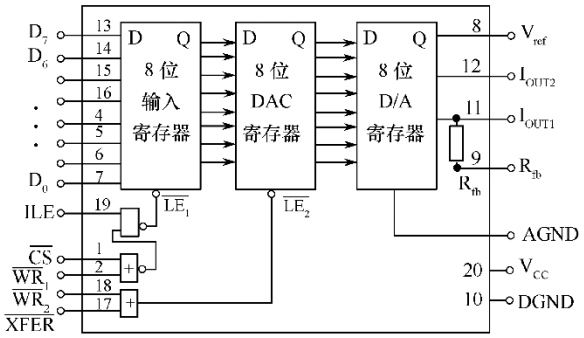


图 7-65　DAC0832 内部逻辑

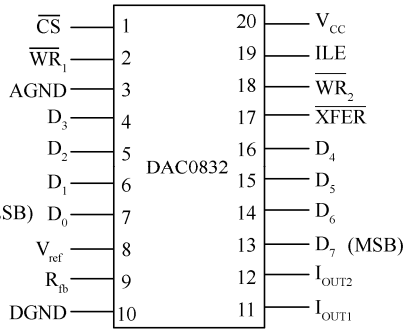


图 7-66　DAC0832 引脚

表 7-7　DAC0832 的引脚功能

引　脚	功　　能	引　脚	功　　能
D_0～D_7	数据输入	V_{CC}	电源输入
ILE	数据允许信号，高电平有效	I_{out1}，I_{out2}	电流输出线，$I_{out1}+I_{out2}$=常数
$\overline{CS}$	输入寄存器选择信号，低电平有效	AGND	模拟信号地
$\overline{WR_1}$	输入寄存器写选通信号，低电平有效	DGND	数字信号地
$\overline{WR_2}$	DAC 寄存器写选通信号，低电平有效	R_{fb}	反馈电阻
$\overline{XFER}$	数据传送信号，低电平有效	V_{ref}	基准电压输入

（3）DAC0832 的工作方式

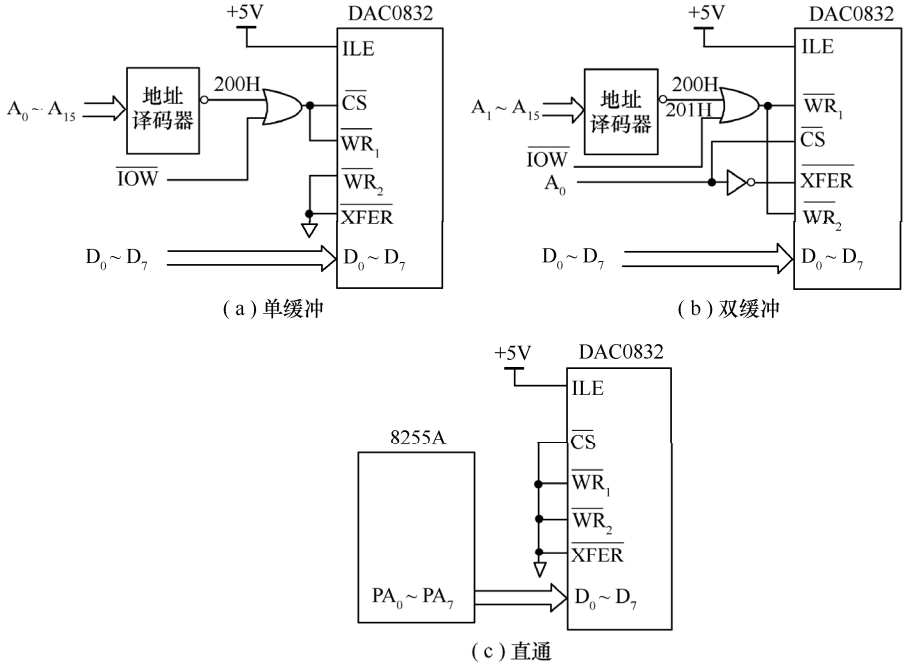
根据输入锁存器和 DAC 寄存器的不同的控制方法，DAC0832 有如下 3 种工作方式。

① 单缓冲方式。控制输入寄存器和 DAC 寄存器同时跟随或锁存数据，或只控制这两个寄存器之一，而另一个接成直通方式。此方式适用于只有一路模拟量输出或几路模拟量非同步输出的情形。参考线路如图 7-67(a)所示，程序中直接写入 D/A 端口即可。

② 双缓冲方式。分别控制输入寄存器和 DAC 寄存器写入，此方式适用于多路 D/A 同时输出的情形，使各路数据分别锁存于各自输入寄存器，然后由相同的写控制信号把数据写入 DAC 寄存器，实现同步转换。参考线路如图 7-67(b)所示，程序段如下：

```
MOV     DX, 200H          ; DAC 0832 输入锁存器的地址为 200H
OUT     DX, AL            ; AL 中数据 DATA 送输入锁存器
```


電子工業出版社



(a) 单缓冲　　(b) 双缓冲

(c) 直通

图 7-67　DAC0832 的工作方式

```
MOV     DX, 201H          ; DAC 0832DAC 锁存器的地址为 201H（A0=1）
OUT     DX, AL            ; 数据 DATA 写入 DAC 锁存器并转换
```

③ 直通方式。输入寄存器和 DAC 寄存器都接成直通方式。此时提供给 DAC 的数据来自外部锁存电路，如图 7-67(c)中 8255A 的 A 口，其程序段如下：

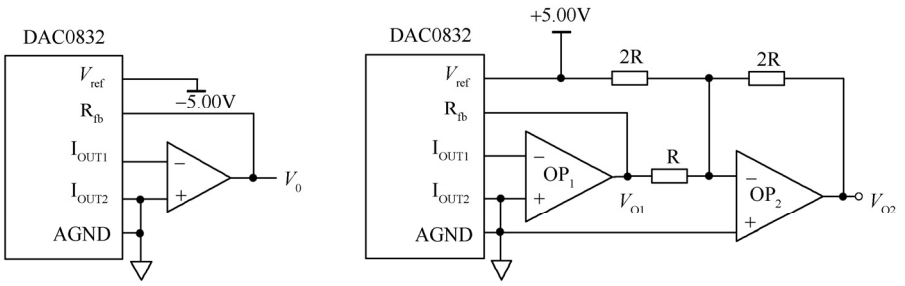
```
MOV     DX, PA8255        ; 设 8255 A 口地址为 PA8255
OUT     DX, AL            ; AL 中数据送 A 口锁存并转换，A 口方式 0，输出
```

（4）DAC 0832 的输出方式

DAC 0832 的输出是电流型的，在微机系统中通常需要电压信号，这时可用运算放大器将其转换为单极性或双极性输出电压。

① 单极性输出。如图 7-68(a)所示，对应数字量 00H～FFH 的模拟电压 V_o 的输出范围是 0～$-V_{ref}$（+5.00 V）。

② 双极性输出。如图 7-68(b)所示，图中单极性输出电压 V_{o1} 经运放 OP_2 电压偏移、放大后，对应数字量 00H～FFH 的模拟电压 V_{o2} 输出范围是$-V_{ref}$～$+V_{ref}$（−5.00～+5.00 V）。



(a) 单极性输出　　(b) 双极性输出

图 7-68　DAC0832 的输出方式

電子工業出版社

4．典型 12 位 DAC 芯片——AD567

考虑到 8 位 CPU 接口的需要，12 位 DAC 一般都采用双缓冲器结构，如 NSC 的 DAC 1210 与 8 位的 DAC0832 结构类似，12 位输入寄存器由 1 个 8 位寄存器和 1 个 4 位寄存器构成，分两次写入后再启动转换，便于实现单片转换或多片同步转换。

美国 ADI 公司的产品 AD567 也是电流输出型高速 12 位 D/A 转换器，双输入缓冲结构，但第一级由 3 个 4 位缓冲器组成，可直接连接 8 位或 16 位数据总线。

（1）AD567 的性能和引脚

AD567 的功能框图如图 7-69 所示，其主要特性如下：

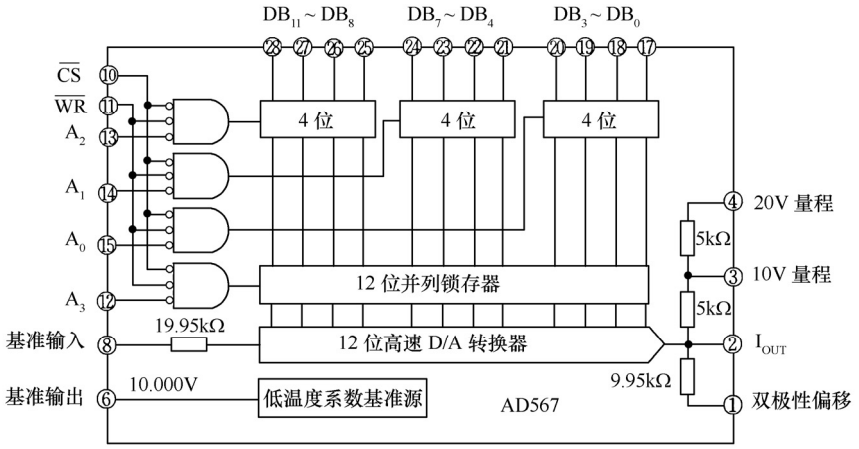


图 7-69　AD567 功能框图

① 单片设计，片内集成高稳定电压基准源，10.000 V±1 mV。

② 与 TTL 和 CMOS 电平兼容。

③ 分辨率 12 位，非线性误差小于±1 LSB。

④ 电流型输出，最大 2 mA，建立时间 500 ns。

⑤ 电源电压范围±12 V～±15 V，低功耗 300 mW。

AD567 的引脚如图 7-70 所示，引脚功能如下：

① DB_{11}～DB_0：12 位数字量输入。

② 引脚 1～4：模拟量输出，支持双极性±2.5 V、±5 V、±10 V 输出，也可以单极性 0～5 V、0～10 V 输出，各种输出电压范围（均对应数字量变化范围 000H～FFFH）的引脚连接如表 7-8 所示，其中±5 V 输出时，模拟信号的参考接线如图 7-71 所示。

③ 引脚 10～15：控制信号，输入锁存器由地址信号 A_3～A_0、片选 $\overline{CS}$ 和写信号 $\overline{WR}$ 控制，真值表如表 7-9 所示。

表 7-8　各种输出范围的引脚连接

输出范围	连接 3 脚到	连接 4 脚到	连接 1 脚到
0～+5V	运放输出端	2 脚	5 脚
0～+10V	运放输出端	运放输出端	5 脚
−2.5V～+2.5V	运放输出端	2 脚	6 脚（串 50Ω 电阻）
−5V～+5V	运放输出端	运放输出端	6 脚（串 50Ω 电阻）
−10V～+10V	悬空	运放输出端	6 脚（串 50 Ω 电阻）

電子工業出版社

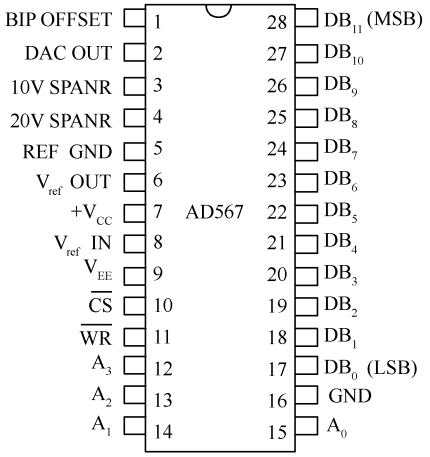


图 7-70　AD567 的引脚图

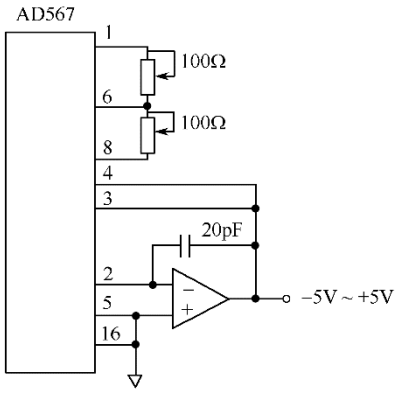


图 7-71　AD567 双极性 ± 5V 输出

表 7-9　AD567 控制真值表

$\overline{CS}$	$\overline{WR}$	A_3	A_2	A_1	A_0	操　作
1	×	×	×	×	×	—
×	1	×	×	×	×	—
0	0	1	1	1	0	A_0=0，锁存第一级缓冲器低 4 位
0	0	1	1	0	1	A_1=0，锁存第一级缓冲器中 4 位
0	0	1	0	1	1	A_2=0，锁存第一级缓冲器高 4 位
0	0	0	1	1	1	A_3=0，锁存第二级缓冲器
0	0	0	0	0	0	所有锁存器均透明

④ 增益微调。在基准输出端（6 脚）与输入端（8 脚）之间接 100 Ω 电位器，通过调节基准电压实现增益微调。

⑤ 模拟地（5 脚）和数字地（16 脚）。在设计时通常把模拟地和数字地只在一点连接，有利于提高输出精度和抗干扰性。

（2）AD567 与 CPU 接口

AD567 和 8 位数据总线连接时，待转换的 12 位数字量至少分两次送出。控制逻辑使它能够使用向左或向右对齐的数据格式（如图 7-72 所示）。

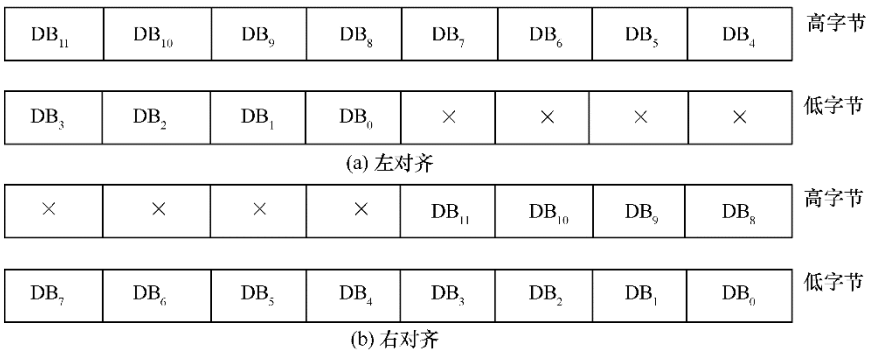


图 7-72　AD567 的 8 位总线与 12 位数据格式

图 7-73 给出了向右对齐的电路连接，这种情况下，设待转换的数据在 AX 中，$\overline{CS}$ 对应

電子工業出版社

的地址为 280H，则输出程序如下：

```
MOV     DX, 280H
OUT     DX, AL          ; A0,A1=0; A2,A3=1，写入第一级缓冲器中 4 位和低 4 位
INC     DX
MOV     AL, AH
OUT     DX, AL          ; 同时打开第一级缓冲器高 4 位（写入原 AH 中低 4 位）
                        ; 和第二级缓冲器（写入第一级缓冲器的 8 位锁存值）
```

AD567 与 16 位总线接口时比较简单，单缓冲即可，图 7-74 给出了一种实现方式。

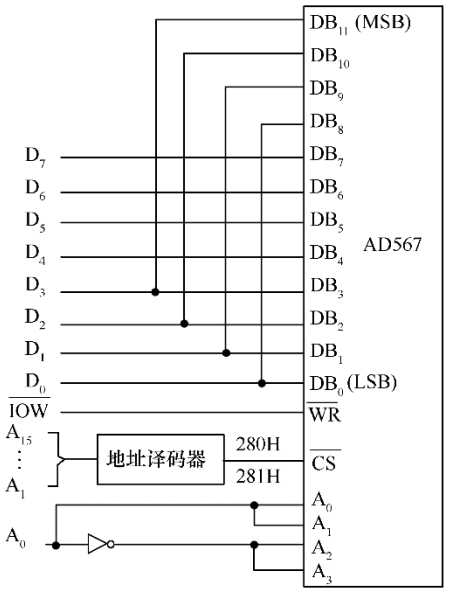


图 7-73　AD567 与 8 位总线接口（右对齐）

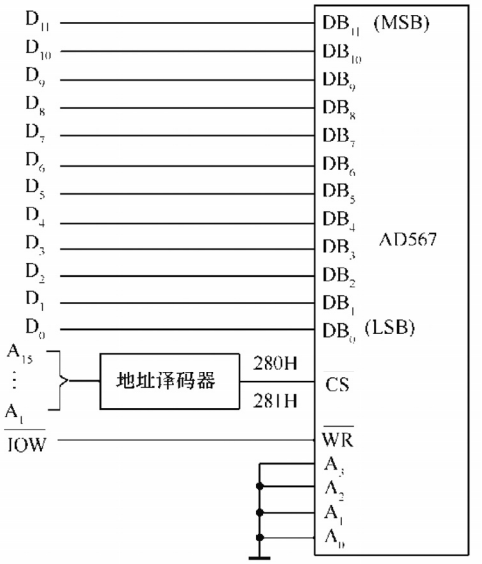


图 7-74　AD567 与 12 位总线接口

5．串行 D/A 转换器

串行 DAC 体积小，接口连线简单，易实现隔离传输，低功耗。在结构上，串行 DAC 有单路和多路之分，分辨率有 8、10、12 位等。典型芯片如 MAX538，它是一个+5 V 供电的单路 12 位低功耗、电压输出的串行 DAC，建立时间 25 μs。内部结构和引脚如图 7-75 所示。引脚定义如表 7-10 所示。

MAX538 使用与 SPI 标准兼容（CPOL=CPHA=0）的三线串联接口。最简单的连线方式是：DAC 的 SCLK 与 SPI 接口的 SCK、DIN 与 MOSI、$\overline{CS}$ 与 I/O 引脚、DOUT 与 MISO 直接相连。

MAX538 DIN 引脚上的数据通过 SCLK 脉冲移位输入至内部 DAC。DAC 需编程写入 2 个 8 位数据块共 16 位，串行数据在时钟作用下移位输入到 DAC。MSB 为首，但前面可填入 4 位（任意数，形成 16 位数，在正常非级联模式下不需输入）。传输过程简单，在片选 $\overline{CS}$ 为低时，数据在每个时钟 SCLK 上升沿输入移位；串行输入数据在 16 位移位寄存器中保持，在 $\overline{CS}$ 信号的上升沿时，最低的 12 位有效位被传送到 DAC 的内部寄存器并更新 DAC 输出。$\overline{CS}$ 高电平时，时钟移位输入不工作。串行接口时序如图 7-76 所示，设计时序时应保证 $t_{CL}>(t_{DO}+t_{DS})$，最大串行时钟速率为 $1/(t_{DO}+t_{DS})$，约 14 MHz，数字量更新速率由片选信号 $\overline{CS}$ 决定，即 $16\times(t_{CH}+t_{CL})+t_{CSW}$，最快约 1.14 μs 或 877 kHz。

電子工業出版社

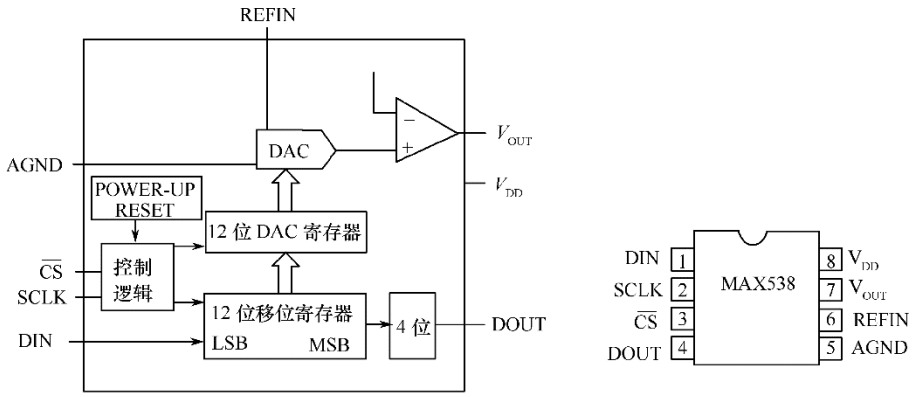


图 7-75　MAX538 芯片内部结构和引脚

表 7-10　MAX538 芯片引脚定义

引脚名	定　义	引脚名	定　义
DIN	串行数据输入	V_{DD}	电源
SCLK	串行时钟输入	AGND	模拟地
$\overline{CS}$	片选	V_{OUT}	电压输出
DOUT	串行数据输出	REFIN	基准电压输入

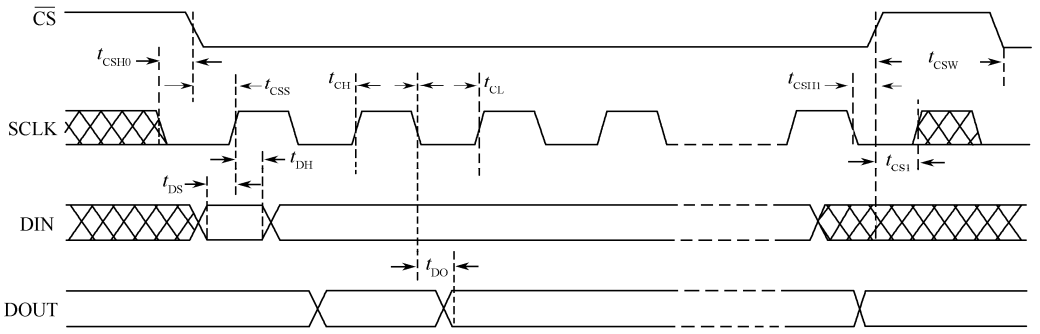


图 7-76　MAX538 串行时序

串行输出引脚 DOUT 允许级联两个或以上 DAC。DIN 上的数据将在延迟 16 个脉冲周期加一个时钟宽度之后出现在 DOUT 引脚上。

内部复位电路保证上电时锁存器复位数值 000H。

7.6.3　并行和串行 A/D 转换器

1．A/D 转换的基本方法和特点

实现模数转换的方法很多，常用的有逐次逼近法、双积分法及电压频率转换法等。

（1）逐次逼近式 A/D 转换器

参考图 7-77，逐次逼近法（或逐次比较法）的主要原理是：将一个待转换的模拟输入信号 V_{IN} 与一个设定信号相比较，根据比较结果决定减小还是增大设定信号，以便逼近输入信号。设定信号由 D/A 转换器的输出 V_o 产生，当与输入的模拟信号“相等”时，得到的 D/A 转换器输入数即为对应的模拟输入的数字量。

電子工業出版社

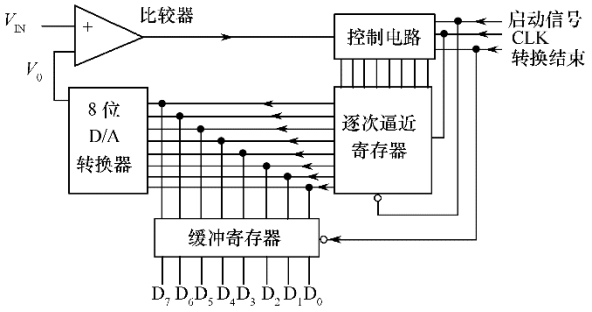


图 7-77　逐次逼近法 A/D 转换器

所谓设定算法是使二进制计数器中的二进制数的每位从最高位起依次尝试置 1，并存入逐次逼近寄存器供DAC输出。每改变一位都要进行比较测试，确定该位的数码应是1还是0。若设定到某位时模拟输入信号 $V_{IN}<V_o$，则比较器的输出为 0，使该位清 0；否则比较器的输出为 1，使该位保持 1。如此逐位设定比较，直到最末位为止。此时保存在 D/A 转换器逐次逼近寄存器的数值即为对应于模拟输入信号的数字量，这就完成了 A/D 转换过程。

（2）双积分法 A/D 转换器

如图 7-78(a)所示，电子开关先把 V_X 采样输入到积分器，积分器从 0 开始进行固定时间 T 的正向积分，时间 T 到后，开关将与 V_X 极性相反的基准电压 V_{ref} 输入到积分器进行反相积分，到输出为 0 V 时停止反相积分。

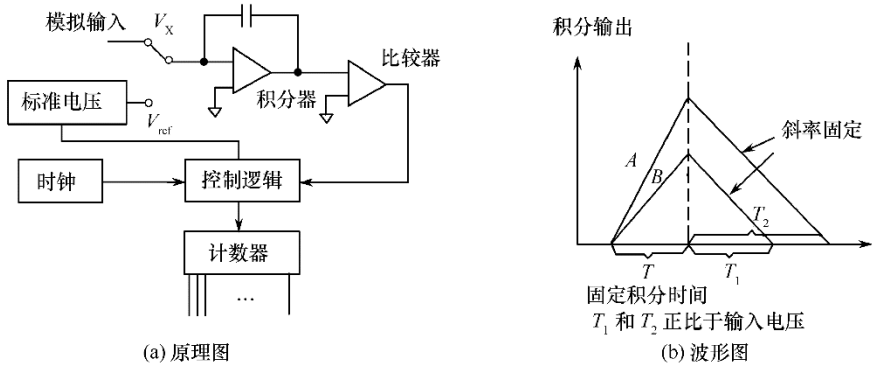


图 7-78　双积分法 A/D 转换原理

由图 7-78(b)所示的积分器输出波形可以看出，反相积分时，积分器的斜率是固定的，V_X 越大，积分器的输出电压越高，反相积分时间也越长。计数器在反相积分时间内的计数值就是与输入电压 V_X 在时间 T 内的平均值对应的数字量。

这种 A/D 的转换速度较慢，但抗高频干扰性能好。

（3）电压频率转换法（Voltage to Frequency，V/F）

当电压 V_X 加至电压频率转换器 VFC 的输入端后，便产生频率 f 与 V_X 成正比的脉冲。由微机对该脉冲计数，每个单位时间内的计数值正比于输入电压 V_X，从而完成模数变换。

VFC 与微处理器结合起来，可方便地构成多位高精度模数转换器，并具有零点漂移及非线性误差等性能优于逐次逼近式 ADC、易于实现远距离或隔离传输、抗扰性好等特点。其缺点是，一般电路多用模拟器件实现，精度不高，且转换速度较慢。

其他模数转换器还有计数式等。

電子工業出版社

2．A/D 转换器的主要性能指标

① 分辨率。表示转换器对微小输入量变化的敏感程度，通常用转换器输出数字量的位数来表示。以 8 位 ADC 为例，分辨率 8 位，数字量变换范围 0～255，当输入电压满量程为 5 V 时，转换电路对输入模拟电压的分辨能力为 5.00 V/255≈19.6 mV。目前常用的 ADC 分辨率有 8、10、12、14 位等。

② 转换时间。指完成一次 A/D 转换所需要的时间，一般为几 μs 至几百 μs。

③ 转换精度。由于模拟量是连续的，而数字量是离散的，所以在某个范围内的模拟量对应于某个数字量。转换精度反映了 A/D 转换器的实际输出范围与理论输出范围接近的程度。通常用数字量的最低有效位（LSB）来表示。

其他性能指标还有对电源电压变化的抑制比、温度系数和增益系数等。

3．典型 8 位 ADC——ADC0809

（1）ADC0809 的性能和结构

ADC0809 是 NSC 公司生产的 CMOS 逐次比较式 A/D 转换器。其主要技术指标如下：

① 分辨率 8 位，模拟电压输入范围 0～5 V。

② 内置集成多路转换开关，8 个模拟输入通道，可在程序控制下对指定通道进行 A/D 转换。

③ 时钟频率 10～1280 kHz，标准时钟频率（640 kHz）下，转换时间为 100 μs。

④ 电源电压+5 V，功耗 15 mW。

ADC0809 的内部结构如图 7-79 所示，引脚信号及功能如图 7-80 和表 7-11 所示。通过引脚 IN_0～IN_7 可输入 8 路模拟电压，每次选通一路转换，其通道号由地址信号（ADD）C、B、A 译码后选定，如表 7-12 所示，片内有地址锁存和译码器。转换结果送入三态输出锁存器，当输出允许信号 OE 有效时，输出到数据总线上。

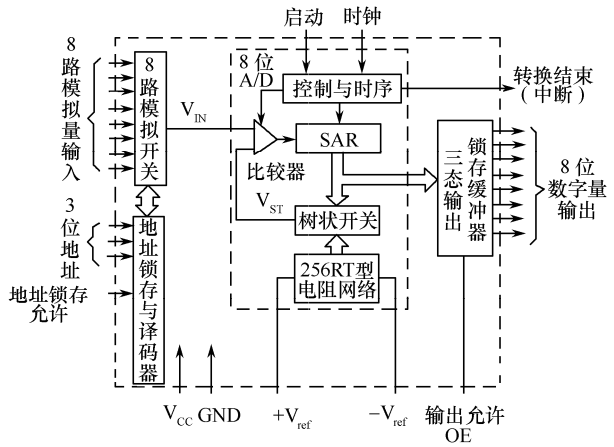


图 7-79　ADC0809 内部结构

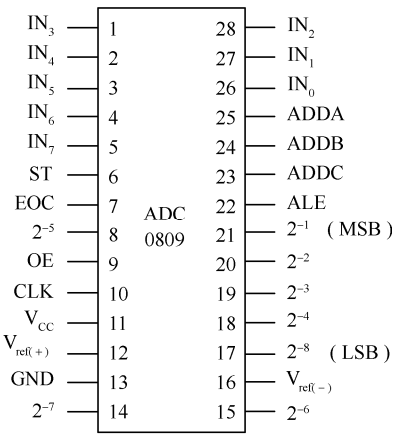


图 7-80　ADC0809 引脚

（2）ADC0809 的转换控制

ADC0809 的转换时序如图 7-81 所示。首先，ADC 通道地址（ADD）C、B、A 在 ALE 的上升沿被锁存；然后，在启动信号 ST（StarT conversion）的下降沿开始转换，EOC 变低；EOC 变高时表示转换结束，这时如 OE 有效，转换结果输出到数据总线。

① 选择通道并启动转换。通常 ST 和 ALE 可由同一正脉冲信号控制，该脉冲的上升沿锁存地址、下降沿启动转换。通道地址可由数据总线输入，如图 7-82(a)所示，以下程序段可

電子工業出版社

表 7-11　ADC0809 引脚功能

引 脚 名	功　能
IN_0～IN_7	模拟电压输入端
（ADD）C，B，A	通道地址信号
ALE	地址锁存信号，上升沿有效
ST	启动转换信号，下降沿有效
EOC	转换结束状态信号，高电平有效
2^{-1}～2^{-8}（D_7～D_0）	数据输出，三态
OE	输出允许信号，高电平有效
CLK	转换定时时钟信号
$V_{ref}(+)$、$V_{ref}(-)$	参考电压输入线，V_{ref}（−）一般为模拟地
Vcc	电源，+5V
GND	数字地

表 7-12　地址与通道号

C　B　A	选中通道
0　0　0	IN_0
0　0　1	IN_1
0　1　0	IN_2
0　1　1	IN_3
1　0　0	IN_4
1　0　1	IN_5
1　1　0	IN_6
1　1　1	IN_7

实现对 IN_3 转换启动：

```
; D2D1D0=通道号，此处为 3
MOV     AL，×××××011B
; 200H～203H 均可
MOV     DX，200H
OUT     DX，AL
```

通道地址也可由地址总线输入，如图 7-82(b)所示，ADC 的通道地址信号（ADD）C、B、A 分别与地址总线的 A_2、A_1、A_0 连接。此时程序中地址号为 DX 的低三位，与 AL 值无关，同样启动对 IN_3 的转换：

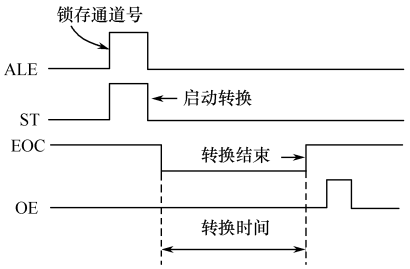
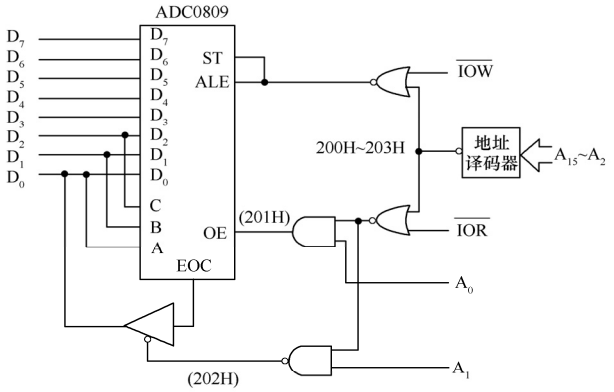
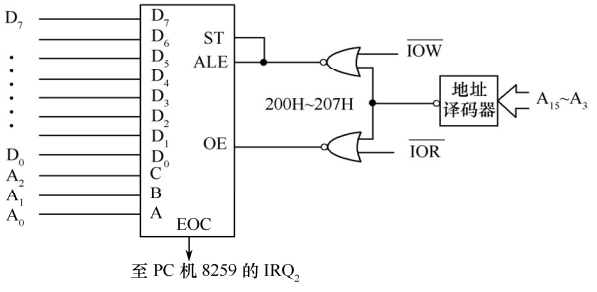


图 7-81　ADC0809 的转换时序图



（a）地址来自数据总线，查询式读取



（b）地址来自数据总线，中断式读取

图 7-82　ADC0809 与系统总线的连接

```
MOV     DX，203H          ; 200H～207H 对应通道 IN0～IN7
OUT     DX，AL            ; AL 值任意，只利用地址译码和 IOW 信号
```


電子工業出版社

② 读取方式：分为直接读取、查询式、中断式三种方式。

直接读取：启动转换后，（软件或硬件）延时一定时间（确保大于 A/D 转换时间）后，直接读取转换结果，不需要利用 EOC 信号。

查询式读取：A/D 转换结束，EOC 由低变高，故查询其值即可知道转换是否结束。如图 7-82(a)所示，EOC 经缓冲器由 D_0 读入 CPU。程序段如下：

```
        MOV     DX, 202H
WAIT:   IN      AL, DX              ; AL 的 D0=EOC
        TEST    AL, 01H
        JZ      WAIT
        …                           ; 读取转换结果
```

中断式读取：A/D 转换结束时也可用 EOC 的上升沿申请中断，CPU 在中断服务程序中读取转换结果。如图 7-82(b)所示，EOC 通过 PC 机 8259 的 IRQ_2 申请中断。

③ 读取转换结果。OE 有效时转换结果数据输出到外部数据线上。如图 7-82 所示接线，读取程序：

```
        MOV     DX, 201H
        IN      AL, DX              ; 转换结果读入 AL 中
```

4．典型 12 位 ADC——AD574/AD1674

AD574 是美国 ADI 公司的产品，是一个完整的 12 位逐次逼近式 A/D 转换器，带有可与 8 位或 16 位 CPU 总线直接连接的三态输出缓冲器。

（1）AD574 的主要特性

① 带有基准源和时钟。

② 带有三态缓冲器输出的 8 位或 16 位微处理器总线接口。

③ 快速逐次逼近式 A/D 转换，转换时间 25 μs（12 位时）。

④ 分辨率 12 位；精度±1 LSB。

（2）AD574 芯片内部结构和引脚

AD574 芯片由两部分构成，一部分为高性能 12 位 D/A 转换器和电源基准，另一部分为低功耗逐次比较寄存器、转换控制电路、时钟、比较器和总线接口。如图 7-83 所示，其引脚共 28 位，DIP 封装，主要信号包括：

① 12 位数字量输出 DB_{11}～DB_0，其中 DB_{11} 为最高位（MSB），DB_0 为最低位（LSB）。

② 模拟量输入 $10V_{IN}$ 及 $20V_{IN}$，可以双极性±5 V 或±10 V 输入，也可以单极性 0～+10 V 或 0～+20 V 输入。

③ 控制信号。AD574 的逻辑控制信号共有 5 个，即 $\overline{CS}$、CE、R/$\overline{C}$、12/$\overline{8}$、A_0。其工作方式如表 7-13 所示。

$\overline{CS}$：片选信号，低电平有效。

CE：芯片启动信号，高电平有效。

R/$\overline{C}$：读出和转换控制信号。

12/$\overline{8}$：数据输出格式选择控制信号。12/$\overline{8}$=1，12 位同时输出；12/$\overline{8}$=0，高 8 位、低 4 位分两次输出，向左对齐数据格式。

電子工業出版社

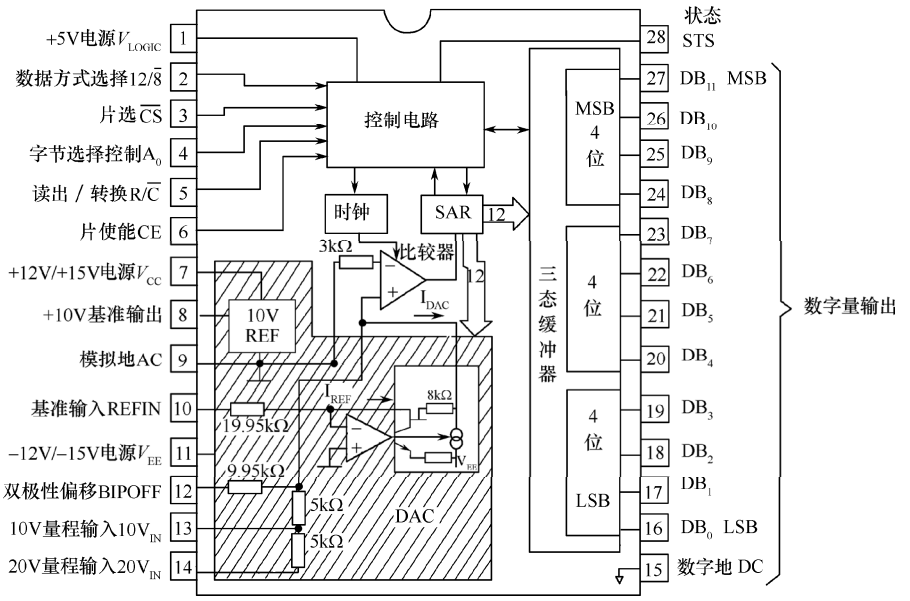


图 7-83　AD574 内部结构和引脚

表 7-13　AD574 工作方式真值表

CE	$\overline{CS}$	R/$\overline{C}$	12/$\overline{8}$	A_0	工作状态
1	0	0	×	0	启动 12 位转换
1	0	0	×	1	启动 8 位转换
1	0	1	1	×	允许 12 位并行输出
1	0	1	0	0	允许高 8 位并行输出
1	0	1	0	1	允许低 4 位加上尾随 4 个 0 输出
×	1	×	×	×	不工作
0	×	×	×	×	不工作

A_0：字节选择控制信号，读或启动时使用。A_0=1，启动 ADC 转换时按 8 位方式。

STS：输出状态信号，高电平有效。在转换过程中，STS 为高电平，转换完成后，该引脚为低电平。

（3）AD574 的升级替代型 AD1674

随着技术的发展，AD674B（15 μs）、AD774B（8 μs）、AD1674（10 μs）等作为 AD574 的高速升级产品进入市场，它们在外部引脚上完全兼容 AD574。AD1674 不仅转换速度明显提升至 10 μs，内部还集成了采样保持器，目前已成为 AD574 的替代升级产品。其内部结构如图 7-84 所示，可见除增加采样保持器和个别参数有所变化（如输入端电阻等）外，原理结构与 AD574 相同。

（4）AD574/AD1674 的输入连接

AD574/1674 的输入连接如图 7-85 所示。各种模拟输入电压范围对应的数字输出范围均为 000H～FFFH（12 位）或 00H～FFH（8 位）。电位器用于零位和满量程调整。

（5）AD574/1674 的输出接口

AD574/1674 与 16 位和 8 位数据总线的连接如图 7-86 所示。与 ADC0809 类似，AD574/1674 也有三种读取方式。直接读取（固定延时）方式实现一次转换的程序段（与 8 位总线接口如图 7-86(b)所示）如下：

電子工業出版社

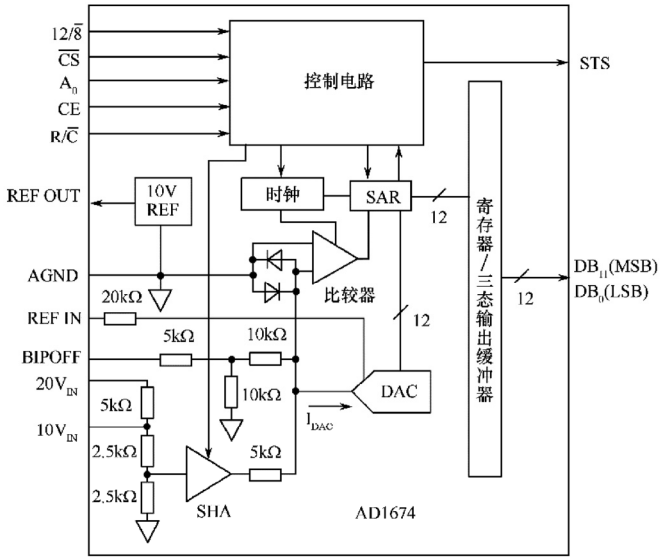
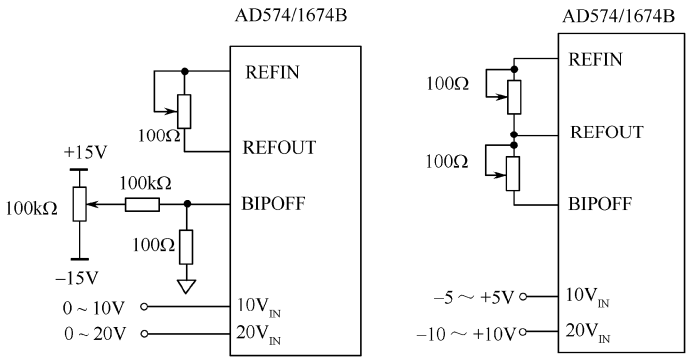
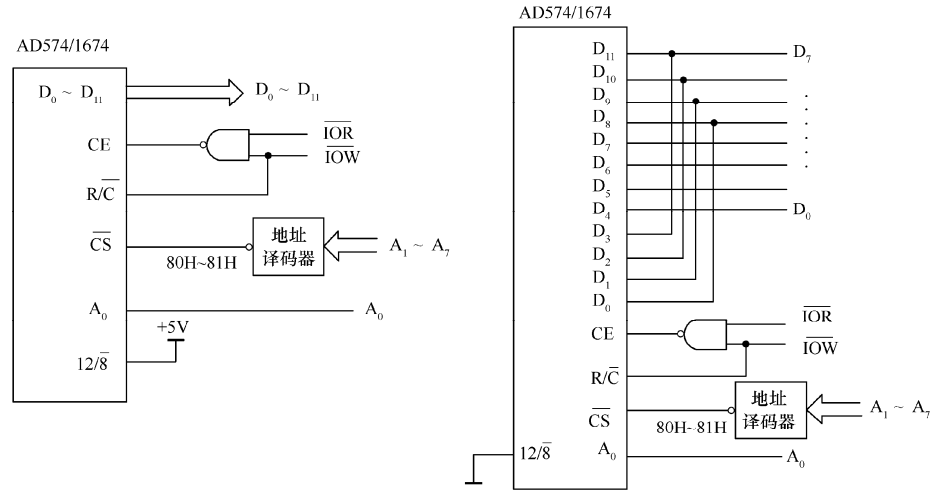


图 7-84　AD1674 内部结构



(a) 单级性输入　　(b) 双级性输入

图 7-85　AD574/AD1674 的输入连接



(a) 与16位总线连接　　(b) 与8位总线连接

图 7-86　AD574/1764 与总线的接口

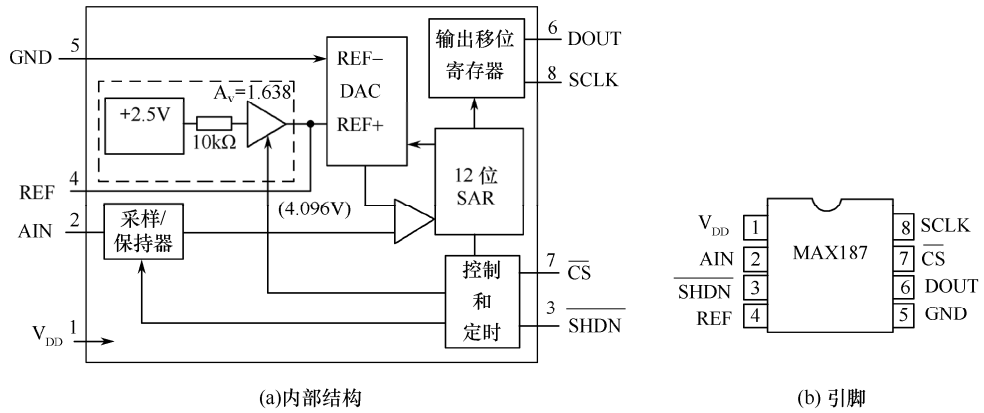
電子工業出版社

```
        OUT     80H, AL                 ; 启动 12 位转换，AL 为任意值
        CALL    DELAY                   ; 延时，等待转换结束
        IN      AL, 80H                 ; 读入高 8 位
        MOV     AH, AL
        IN      AL, 81H                 ; 读入低 4 位及 4 位 0
```

5．典型串行 ADC 芯片——MAX187

MAXIM 公司的串行 ADC 产品 MAX187 的主要性能指标是：分辨率 12 位；非线性误差±1/2 LSB；片内集成采样/保持器，快速采样/保持（1.5 μs）、转换（8.5 μs）；单电源+5 V±5%，低功耗；片内基准源 4.096 V；三线制串行接口，兼容 SPI 等总线标准。

MAX187 内部结构框图和引脚信号如图 7-87 所示，具体引脚功能如下。



(a)内部结构　　(b) 引脚

图 7-87　串行 ADC MAX187 内部结构和引脚

V_{DD}：电源电压，5 V±5%。

AIN：模拟电压输入，−0.3 V～V_{DD}+0.3 V。

$\overline{SHDN}$：为高电平，则 ADC 处于工作状态；为低电平，则 ADC 处于休眠状态（1 μA）。

V_{ref}：基准电压，片内 4.096 V（即数字输出 000H～0FFFH 对应模拟输入 0～4.096 V）。

SCLK：串行时钟输入，最高频率 5 MHz。

$\overline{CS}$：片选信号。下降沿启动转换。$\overline{CS}$=1 时，DOUT 处于高阻状态

DOUT：串行数据输出。数据在 SCLK 的下降沿变化。

GND：模拟和数字地。

与 SPI 接口连接时 $\overline{CS}$ 接 SPI 的 I/O 引脚，SCLK 接 SCK，DOUT 接 MISO，SPI 的 $\overline{SS}$ 接上拉电阻至+5 V。

转换时序如图 7-88 所示。令 $\overline{CS}$ 变低，开始转换、DOUT 变低。待 DOUT 变高，则表示转换结束之后在每一个 SCLK 的上升沿可读入一位数据（最高位在前）。D_0（LSB）读入后，再有 SCLK 脉冲，则读入数据为 0。

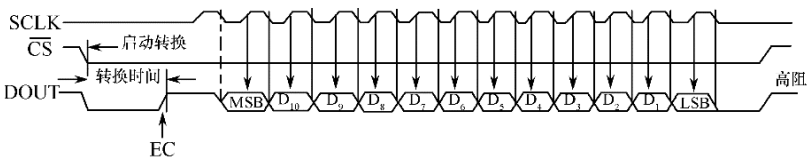


图 7-88　串行 MAX187 转换时序

電子工業出版社

习 题 7

7-1　要使用可编程定时/计数器 8253 的 0 通道周期性地发出脉冲，周期为 1 ms，试编写初始化程序（地址自定，输入脉冲频率= 2 MHz）。若要求 8253 的端口地址为 80H～83H，试用 3-8 译码器实现地址译码，并完成 8253 芯片与 PC 总线之间其他信号的连接。

7-2　用 8253 芯片组成一个实时时钟系统。0 通道作为秒计时器，1 和 2 通道作为计数器，分别用作分、时计时，试画出硬件电路并编写主程序和中断服务程序（设系统频率已分频为 50 kHz）。

7-3　假定一片 8253 芯片连接一个 1 kHz 的时钟，用该 8253 以 BCD 格式保持一天中的时间，精度为秒。在 HOURS（小时）、MINUTES（分）、SECOND（秒）和 AMPM（上、下午）等字节均装入当前时间以后，就立即开始计时。编写一个 8253 芯片的初始化程序和一个在每秒结束时修改时间存储单元的中断例程。

7-4　试利用 IBM PC 系统上的 8253（计数器 0）产生年、月、日、时、分、秒的时间记录，并显示在显示器上。在键盘上输入当前的月、日、时、分后启动 8253 的计时操作。

7-5　试利用 8253 芯片设计一个脉冲波群发生器。该发生器周期性地输出 500 kHz、200 kHz、100 kHz、50 kHz、20 kHz、10 kHz、5kHz、2 kHz、1 kHz 的方波，每种频率的信号都持续 10 ms。假定可提供给 8253 的时钟频率为 2 MHz，8253 的端口地址为 2C0H～2C3H。试完成硬件和软件设计。

7-6　试说明如何利用 8253 芯片测量从同一信号线送来的两个脉冲的时间间隔，测量的最大时间间隔为 1 小时，读时精度为 1 ms。试完成硬件和软件设计，8253 输入时钟频率 1 MHz，端口地址 2A0H。

7-7　设 8255A 的端口地址为 60H～63H，试编写下列情况的初始化程序：

（1）将 A 组和 B 组设置为方式 0，端口 A 和 C 作为输入口，端口 B 作为输出口。

（2）将 A 组设置为方式 2，B 组设计为方式 1，端口 B 作为输出口。

（3）将 A 组设置为方式 1，且端口 A 作为输入，PC_6 和 PC_7 作为输出；B 组设置为方式 1 且端口 B 为输入口。

7-8　有一输入设备，当有一个数据要输入计算机时，就发出一个数据准备就绪的状态信号 READY（高电平有效），当 CPU 把这个数据取走时，清除 READY 信号，使其变为低电平。试用 8255 芯片作为接口电路，分别用查询和中断方式从外设读入 100 字节数据，存放在从 DATA 开始的内存单元中，画出硬件电路图，编出程序。

7-9　图 7-89 为一报警系统，当碰断微型导线时，喇叭将发出报警信号。试编写 8255 初始化程序和中断服务程序（提示：PC_0 以一定频率输出 0101……即脉冲，周期自定）。

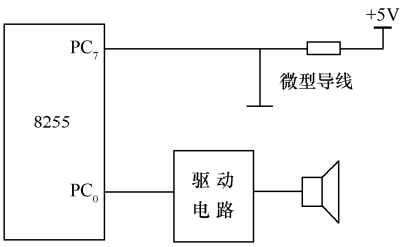


图 7-89　习题 7-9 图

7-10　设计一个交通信号灯自动控制系统。设计要求如下：

（1）在某一南北方向和东西方向的十字路口，每个方向都有红、黄、绿三色交通信号灯，如图 7-90 所示。要求信号灯按下述规则变化：

首先使某一方向绿灯亮，另一方向红灯亮，当出现下述情况之一时，开始变灯：① 红灯已亮 30 秒钟；② 红灯路口保留的车辆已达 5 辆。

变灯的方法如下：首先使绿灯闪烁 5 秒钟（0.5 秒灭，0.5 秒亮，重复 5 次），接着绿灯变为黄灯，再经过 5 秒钟。这期间，另一方向的红灯保持不变，然后使黄灯变为红灯，同时使另一方向

電子工業出版社

红灯变为绿灯。

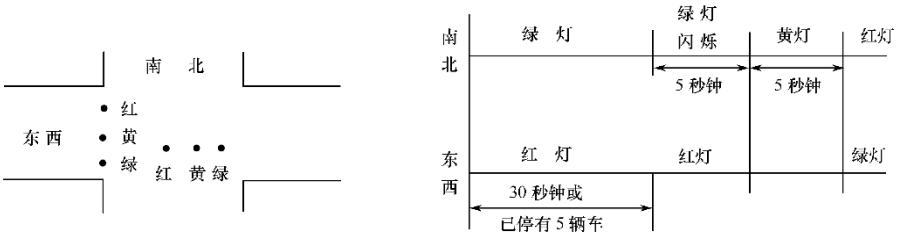


图 7-90　习题 7-10 图

（2）画出硬件框图，编写控制程序（提示：用 8255 某一端口控制 6 个交通灯，用 8253 芯片实现定时和检测车辆）。

7-11　如图 7-91 所示为开关状态检测和继电器控制电路。当开关闭合时，将驱动对应的继电器（有驱动电流流过继电器线圈）；若某开关处于断开状态，则无电流流过继电器线圈。若系统每隔 20 ms 检测一次开关状态，对继电器做出相应控制，定时控制由 8253 芯片实现，试完成对 8255A 芯片的初始化程序（初始状态所有继电器的线圈都无电流流过）和检测控制程序。设 8255A 端口地址为 3C0H～3C3H。

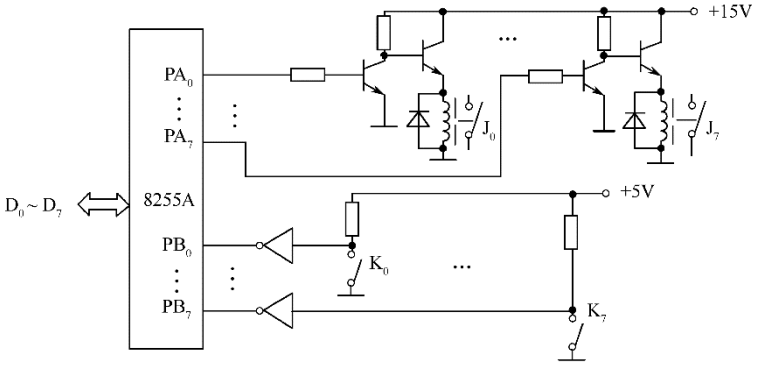


图 7-91　习题 7-11 图

7-12　试利用 8255A 芯片设计一个事件统计与显示电路。当某事件出现（如生产流水线上的工件检测器检测到有一个工件通过）时，送来一个负脉冲。显示器由两位 7 段 LED 显示器组成，初始状态显示 00。之后，每送来一个负脉冲，则显示内容加 1，当显示到 99 时，若再送来一个负脉冲，则又显示 00，同时通过 8255A 的端口引脚输出一个正脉冲，用于控制声、光电路。设 8255A 的端口地址为 03C0H～03C3H，试完成硬件和软件设计（不包括声、光电路）。

7-13　若 8251 芯片的主参考时钟频率为 1.2288 MHz，其 BAUDOUT 和 RCLK 相连，试完成以下初始化程序设计：半双工通信，每字符的数据位数为 7 位，停止位为 1 位，偶校验位，波特率为 1200 B/s，处于发送状态。8251 芯片的数据端口地址为 2C0H，控制端口为 2C1H。

7-14　若两台计算机利用 8251 进行串行通信。通信规程如下：异步传送，波特率为 2400 b/s，字符格式数据为 8 位，1 位停止位，无校验。试完成下列程序设计（可自行指定端口符号地址）：

（1）发送程序。将数据段中以 T_BUF 为起始地址的一组数据发送到接收方，字符串长度存于字节变量 T_LEN 中。发送时，先发 2 个起始符 AAH 和 55H。发送完全部数据之后，再发送 55H 和 AAH 作为发送结束标志。

（2）接收程序。将接收到的一组数据存放在从 R_BUF 开始的接收缓冲区中，所收到的数据长

電子工業出版社

度存于字节变量 R_LEN 中，只统计数据个数。若接收正确，将字节变量 ERROR 置为 00H，否则置为 FFH。报文也以 AAH 和 55H 开始，以 55H 和 AAH 结束。

7-15　试设计一个自检程序，检验 IBM PC 的异步通信卡中 8251 是否正常工作。该检验程序从键盘接受输入并将输入的字符发送出去，接收的字符在显示器上显示出来。假定 8251 芯片已初始化，数据端口地址 DATA8251，控制端口 CTRL8251。

7-16　试编写一段用 8237DMA 控制器执行存储器到存储器数据传送的程序，把从 SOURCE 开始的 1000 字节传送到从 DST 开始的数据区，8237 的端口地址为 0600H～060FH。

7-17　试利用 8253、8255A、AD574 芯片设计一个数据采集系统（不包括 A/D 转换器输入通道中的放大器和采样/保持电路）。要求每隔 50 μs 采集一个数据，数据的 I/O 传送控制采用中断控制，8255A 的 INTR 信号接至 8259A 的 IR_2 请求信号引脚。允许附加必要的门电路或单稳态电路。试完成：

（1）硬件设计，说明原理，画出连接图（不包括 8259）。

（2）软件设计，说明原理，编写程序包括 8255A、8253 的初始化及中断服务程序。

7-18　已知模拟量输入信号为 0～+5 V，试设计利用 ADC0809 芯片进行查询式 A/D 转换的 PC 接口卡，并编写实现一次采集的程序段。

7-19　在 PC 总线上扩充 DAC0832 芯片，并完成四路三角波信号输出。要求：

（1）三角波的电压范围为 0～+2.5 V。

（2）画出硬件接线图，DAC0832 可用地址：280H～283H。

（3）编写 D/A 转换程序。

7-20　要求将内存单元 BUF1 和 BUF2 中的数据同时转换为模拟电压输出，采用 DAC0832 实现，设计有关硬件和软件，说明设计原理，画出电路原理图。

7-21　收集相关资料，画出一基于 USB 总线的数字量、模拟量或通信转换电路原理图。

電子工業出版社

第 8 章　微型计算机系统的发展

8.1　微型计算机体系结构及系统总线

8.1.1　微型计算机体系结构

目前，市场上的主流微型计算机是以 Intel 公司的 80x86 系列 CPU 为核心的个人计算机，通常称为 PC（个人计算机）系列机。PC 系列机经历了 IBM PC、PC/XT、PC/AT 及其兼容机以及 80386、80486、Pentium 和 Core（酷睿）等发展阶段。当前的微机系统大多采用以 Core i5、Core i7 为核心的 CPU，其运算速度和性能有了很大提高，体系结构也有了很大发展。

1．IBM PC/AT 微机系统

IBM PC/AT 机采用 80286 CPU，它兼容 8086/8088 指令系统，支持虚拟存储和多任务操作，对存储器的访问有两种方式：实地址和保护虚地址模式。

PC/AT 机的微处理器子系统由 80286 微处理器、80287 数值运算协处理器、80284 时钟发生器及其他支持芯片组成，仍用通用的地址锁存器和双向数据收发器形成 24 位地址总线及 16 位数据总线，控制总线由专用的 82288 总线控制器管理，DMA 控制器 8237 和中断控制器 8259A 与 PC/XT 机相比均扩展为 2 片。PC/AT 微机使用 ISA 系统总线，该系统的组成结构如图 8-1 所示。

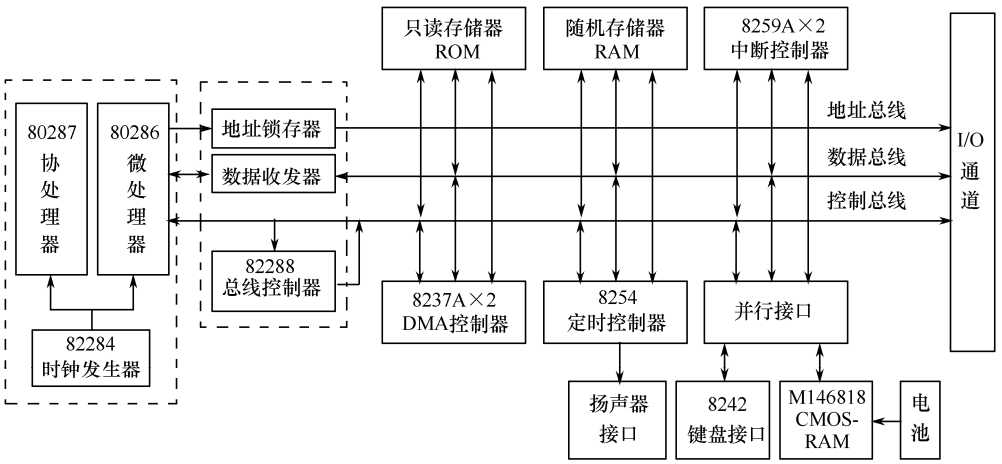


图 8-1　PC/AT 机组成结构

在 PC/AT 机中，内存最低端的 640 KB（000000H～09FFFFH）属于基本 RAM 区；接下来的 128 KB （0A0000H～0BFFFFH）由显示卡提供，用于显示缓冲；接下来的 256 KB（0C0000H～0FFFFFH）为 ROM 区，其低端的 128 KB 分配给 I/O 通道插卡使用，接下来的 64 KB 由 ROM BIOS 占用，最后 64 KB 由系统保留。以上 1 MB 被称为常规内存，其空间分配与 PC/XT 基本相同。由 100000H 向上的 15 MB 是 PC/AT 新增的内存空间，被称为扩展内

電子工業出版社

存。其高端 128 KB 为 ROM 区，保留给以后的系统使用，也可将系统的 ROM BIOS 重新安排到这里；其余 RAM 区，可由用户使用。

PC/AT 兼容机中有一部分已经采用了 CHIPS AND TECHNOLOGIES 公司推出的 CHIPS 门阵电路。PC/AT 兼容机采用的门阵电路主要有 82C201 系统控制芯片、82C202 地址译码芯片、82C203 总线形成芯片、82A204 地址总线形成和刷新地址计数器芯片以及 82A205 数据总线形成和奇偶校验芯片等 5 种。这些芯片同样也实现了 I/O 接口电路和 CPU 的其他支持芯片的功能，并且使系统主板结构更加简洁、可靠。

2．80386、80486 微机系统

就硬件的基本组成来说，80386、80486 微机系统仍然与 PC/XT 及 PC/AT 微机系统类似。自 80386 CPU 问世以来，很多厂商纷纷推出各种牌号的 80386 微机，这些微机最核心的芯片——80386 微处理器和 80387 数学协处理器是一致的，外围芯片则各不相同。以 80386 为基础的微机系统由 80386 CPU、数学协处理器 80387、外部设备控制器 82380 和高速缓冲存储器 Cache 控制器 82385 组成。典型 80386 微机系统组成结构如图 8-2 所示。

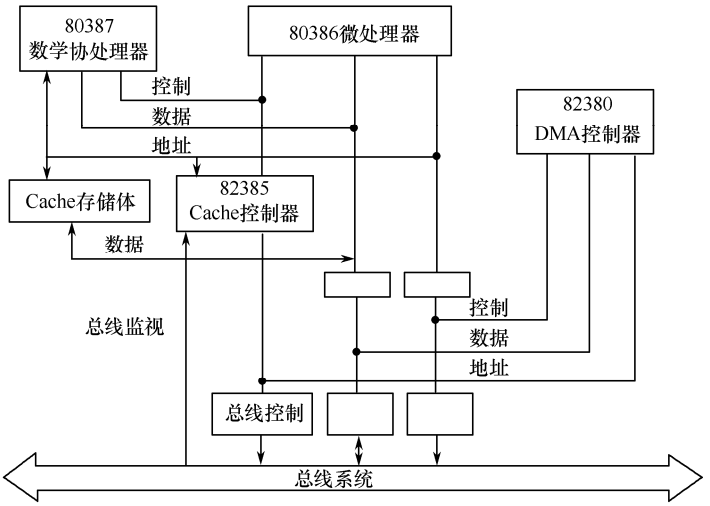


图 8-2　80386 微机系统组成结构

外部设备控制器 82380 是整个微机系统的核心，集成了多个不同功能的接口组件，包括 32 位 8 通道 DMA 控制器、20 级可编程中断控制器、4 个 16 位可编程区间计时器、系统复位逻辑、可编程等待状态控制器、DRAM 刷新控制器以及内部总路线仲裁和控制器等。

其中，DMA 控制器可以减少对微处理机的中断次数，加快多页传送的速度，为需要分页的环境提供极佳的性能；可编程中断控制器（简称 82380 PIC）由三个增强功能的 8259A 中断控制器组成，这 3 个中断控制器提供 15 个外部和 5 个内部中断请求输入，每个外部请求输入可以再级联一个 8259A 从控制器；4 个 16 位可编程区间计时器与 8253 可编程计时器功能相同，尽管这 4 个计时器共享一个时钟输入，但每个计时器都能在 6 种方式的任一种方式下操作，而且这个时钟输入可以不受系统时钟的约束；82380 还有一个专门的系统复位逻辑，通过软件复位和时钟发生器 82384 的硬件复位信号就能启动复位功能。

80486 微机和 80386 微机的硬件结构大体相同，在系统配置上，80486 的规模更大一些，如内存容量通常为 4～8 MB，硬盘容量可达 160～500 MB。

電子工業出版社

3．Pentium 微机系统

Pentium CPU 相比 80486 CPU 在体系结构等方面有了很大的改进。这些改进包括优化的 Cache 结构、更宽的数据总线、更快的协处理器、双整型处理器以及分支预测逻辑，从而显著地提高了处理速度。从 Pentium 微机开始，微机系统通过芯片组协调和控制数据在 CPU、内存和各部件之间的传输。芯片组的型号往往决定了系统的主要性能，包括 CPU 类型、最高工作频率、内存的最大容量和扩展槽的数量等。在 Pentium 时代的早期，芯片组由 4 片组成（430/440/450 系列芯片组等），随着集成电路技术的发展，芯片组中的芯片缩减到 2 片，分别为北桥和南桥芯片。当前主流 Pentium 系列微机系统的组成结构如图 8-3 所示。

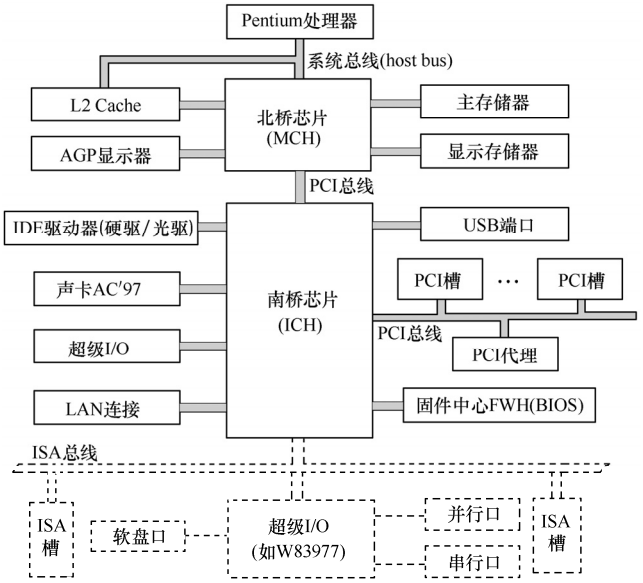


图 8-3　Pentium 系列微机系统组成结构

北桥芯片直接与 Pentium 处理器总线（系统总线）相连，主要控制主存、显存和 AGP/PCI 显示器，所以又称为存储器控制中心（MCH）或图形存储器控制中心（GMCH）。当然，它除了提供对存储器和图形显示的控制外，还提供电源管理和 ECC 数据纠错等功能。若处理器内未含 L2 Cache，它还会提供对 L2 Cache 的控制。相对于南桥芯片，北桥芯片起着主导作用，所以被称为主桥。它主要决定主板的规格、对硬件的支持以及系统的性能。

南桥芯片不与 CPU 总线直接相连，而通过带宽为 166MHz 的 PCI 总线或带宽为 266MHz 以上的专用新型高速总线与北桥芯片相连。南桥芯片控制若干 USB 接口和 PCI 插槽，并通过快速 IDE 接口控制硬盘和光盘驱动器，通过超级 I/O 接口控制键盘、鼠标、打印机和软驱等外设，同时控制声卡、LAN 和 BIOS 固件等。有的芯片组的南桥芯片还支持形成 ISA 扩展总线，如图 8-4 中下侧虚线所示。这样，微机系统可以外接一些 ISA 总线设备（注：现在的 Pentium 4 微机一般不提供 ISA 总线扩展槽）。可见，南桥芯片主要控制主板上的各种接口、PCI 总线、IDE 以及主板上的其他芯片等，所以又称它为 I/O 控制中心（ICH）。

设计、生产控制芯片组的公司很多，目前主流 Pentium 芯片组的生产厂商主要有 Intel、AMD、VIA（威盛）、SiS（矽统）和 ALI（扬智）等公司。其中，Intel 和 AMD 公司也是微处理器主要生产厂商，他们设计的芯片组性能优异，但一般只与自己的微处理器产品配套，而不兼容其他厂商的微处理器，甚至完全排斥异种配置。

電子工業出版社

4．Core 微机系统

Intel 继 Pentium 处理器之后推出 Core 2 Duo 和 Core 2 Quad 品牌，以及最新出的 Core i7、Core i5 和 Core i3 三个品牌的 CPU。以 Core i5 为例，它集成了双通道 DDR3 存储器控制器、PCI-Express 控制器以及部分北桥的功能。Core i5 采用 45 nm 制程，有四个核心，不支持超线程技术，共提供 4 个线程方面，每个核心拥有各自独立的 256 KB 的 L2 Cache，并且共享 8 MB 的 L3 Cache。Core i5 采用 Intel P55 芯片组，P55 会采用单芯片设计，功能与传统的南桥相似。当前主流 Core i5 微机系统组成结构如图 8-4 所示。

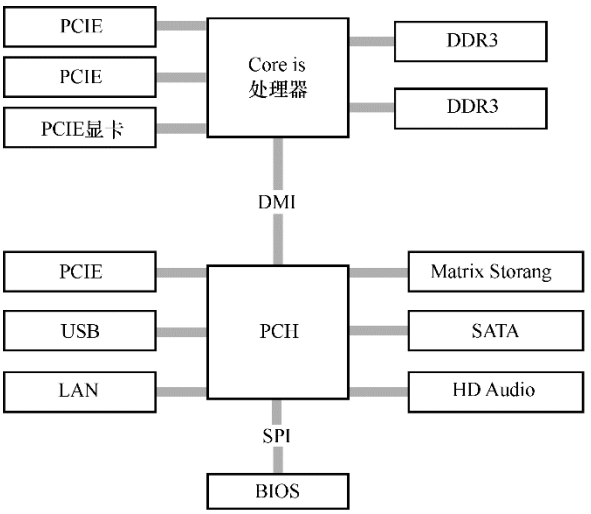


图 8-4　Core i5 微机系统组成结构

Intel 主板芯片组一直采用 MCH（北桥芯片）加 ICH（南桥芯片）的组合。其中北桥芯片主要负责与 CPU、内存、PCI-Express 之间的数据沟通，南桥则主要负责对磁盘及外围设备的管理。随着自身 CPU 架构的改变，Intel 在 5 系列的主流主板芯片组上也将采用单芯片设计。P55 上的那颗主控芯片既不叫北桥，也不叫南桥，而是叫做 PCH 芯片。这颗 PCH 芯片主要负责 LAN 的管理、I/O 设备的管理等工作。而内存方面的控制则交由 CPU 来负责。P55 使用的 DMI(Direct Media Interface)总线与 Core i5 连接，带宽为 2GB/s。

8.1.2　系统外部总线

前面已了解到，总线是微机中各模块之间传送信息的通道，各模块分时共享总线。为了方便微机系统的扩展及各厂商产品的互换和互连，国际上根据微机的发展制定了许多总线标准，如 PC 总线标准、ISA 总线标准和 PCI 总线标准等。

1．ISA 总线

XT 总线仅适合 8 位数据传输，所以从 IBM PC/AT 微机开始采用 ISA 总线（也称为 AT 总线），以便进行 16 位数据传输。XT 总线也被称为 8 位 ISA 总线，ISA 总线为 16 位 ISA 总线。ISA 总线的数据传输速率最高为 8 MB/s，地址总线宽度为 24 位，可支持 16 MB 内存。为了与 XT 总线兼容，在 XT 总线的基础上，ISA 总线延伸出一小段插槽，如图 8-5 所示。

这小段插槽上的引线主要是扩展的地址线和高 8 位数据线及增加的中断申请线等。对于

電子工業出版社

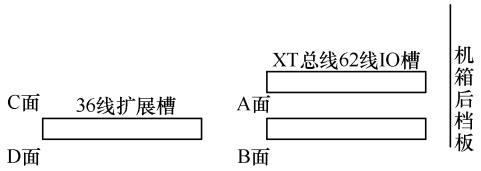


图 8-5　XT 总线和 ISA 总线插槽

早期的 XT 总线扩展卡，若插入 ISA 总线插槽，只使用右边较长的插槽，这部分和原 XT 总线基本一样，仅两处做了改动：① 原 B_{19} 为 $\overline{DACK\,0}$，现因 AT 机的 DRAM 刷新不再通过 DMA 伪传输完成，故直接由系统板上刷新电路产生 $\overline{REFRESH}$ 信号代替（输出）；② 原 B_8 为 $\overline{CARD\ SLCTD}$，现引入 $\overline{OWS}$（零等待状态）信号，表示接口卡上的设备不需插入任何附加等待状态即可完成当前总线周期。ISA 总线的引脚名称及功能如表 8-1 所示。

表 8-1　ISA 总线引脚功能

引　脚	名　称	功　能
D_7～D_0	数据总线低 8 位引脚	8 位数据线，双向，三态；对于 16 位 ISA 总线，是数据线的低 8 位
A_{19}～A_0	地址线引脚	输出 20 位地址信号
$\overline{SMEMR}$ 、$\overline{SMEMW}$	存储器读、写命令引脚	输出存储器读、写命令信号，低电平有效
$\overline{IOR}$ 、$\overline{IOW}$	I/O 读、写命令引脚	I/O 读、写命令，输出，低电平有效
AEN	地址允许信号引脚	输出地址允许信号，高电平有效。该信号由 DMAC 发出，为高表示 DMAC 正在控制系统总线进行 DMA 传送，所以可用于指示 DMA 总线周期
BALE	总线地址锁存允许信号引脚	输出总线地址锁存允许信号。该信号在 CPU 总线周期的 Tl 期间有效，可作为 CPU 总线周期的指示
I/O CHRAY	I/O 通道是否准备好引脚	输入 I/O 通道是否准备好信号，高电平有效。该信号与 8086 的 READY 功能相同，用于插入等待时钟周期
IRQ_7～IRQ_2	中断请求信号引脚	6 个中断请求信号，输入，这些信号由低到高的跳变表示中断请求，但应一直保持高电平，直到 CPU 响应中断为止
DRQ_3～DRQ_1	DMA 请求信号引脚	3 个 DMA 请求信号，输入，高电平有效，分别接到 DMA 控制器
$\overline{DACK_3}$ ～ $\overline{DACK_1}$	DMA 响应信号引脚	3 个 DMA 响应信号，输出，低电平有效
T/C	计数结束信号引脚	计数结束信号，输出，高电平有效。它由 DMAC 发出，用于表示进行 DMA 传送的通道编程时规定传送字节数已经传送完
OSC	输出脉冲引脚	输出振荡器的脉冲
CLK	系统时钟信号引脚	系统时钟信号，输出
RESET	系统复位信号引脚	系统复位信号，输出，高电平有效；该信号有效时表示系统正处于复位状态，可利用该信号复位总线板卡上的有关电路
NOWS	零等待状态信号引脚	输入零等待状态信号，低电平有效，用于缩短按照缺省设置应等待的时钟数；当它有效时，不再插入等待时钟
$\overline{REFRESH}$	刷新信号引脚	刷新信号，双向，低电平有效，由总线主控器的刷新逻辑产生。该信号有效表示存储器正处于刷新周期
SD_{15}～SD_8	数据总线高 8 位引脚	数据总线的高 8 位，双向，三态
$\overline{SBHE}$	总线高字节传送允许信号引脚	高字节允许信号，低电平时表示数据总线正在传送高字节 SD_{15}～SD_8，16 位设备可以利用 $\overline{SBHE}$ 控制 SD_{15}～SD_8 接到数据总线缓冲器上
LA_{23}～LA_{17}	非锁存的地址总线引脚	非锁存的地址线，在 BALE 为高电平时有效。将它们锁存起来，并和已锁存的低位地址线 A_{19}～A_0 组合在一起，可形成 24 位地址线，因而使系统的寻址能力扩大到 16 MB
$\overline{MEMR}$ 、$\overline{MEMW}$	存储器读、写信号引脚	存储器读、写信号，低电平有效。这两个信号在所有的存储器读或写周期有效。$\overline{SMEMR}$ 和 $\overline{SMEMW}$ 仅当访问存储器的低 1 MB 时才有
$\overline{MEMCS16}$	存储器片选 16 引脚	如果总线上某一存储器卡要传送 16 位数据，则必须产生一个有效的（低电平）MEMCS16 信号，该信号加到系统板上，通知主板实现 16 位数据传送。该信号需利用三态门或集电极开路门驱动
$\overline{I/OCS16}$	I/O 片选 16 引脚	与 $\overline{MEMCS16}$ 类似，如果某一 I/O 接口卡要传送 16 位数据，则必须产生有效的（低电平）$\overline{I/OCS16}$ 信号，该信号加到系统板上，通知主板实现 16 位数据的传送。该信号也需利用三态门或集电极开路门驱动

电子工业出版社

续表

引　脚	名　称	功　能
$\overline{\text{MASTER}}$	总线主控信号引脚	该信号与 DRQ 线一起用于获取对系统总线的控制权，使 I/O 通道上的处理器暂时控制系统总线并访问存储器和外设
IRQ_{15}～IRQ_{10}	中断请求信号引脚	可屏蔽中断请求信号
DRQ_7~DRQ_5、DRQ_0	DMA 请求信号引脚	通道 7～5、通道 0 的 DMA 请求信号
$\overline{DACK_7}$ ～ $\overline{DACK_5}$ $\overline{DACK_0}$	DMA 响应信号引脚	通道 7～5、通道 0 的 DMA 响应信号

2．PCI 局部总线

PCI（Peripheral Component Interconnect，外围部件互连）总线是 1992 年以 Intel 公司为首的集团厂家设计的一种先进的高性能局部总线。它支持突发读/写和并发工作方式，并支持多个主控设备；支持的数据线为 32 位（此时最高数据传输速率达 133 MB/s），可扩充到 64 位（此时最高数据传输速率为 264 MB/s）；与 CPU 异步工作，总线工作频率固定为 33 MHz，与 CPU 的工作频率无关，可适合各种不同类型和频率的 CPU，再加上 PCI 支持 3.3 V 电压操作，不但可用于台式机，而且可用于便携机、服务器和一些工作站中；支持多主控设备和并发工作，还支持无限读/写突发方式；具有即插即用（Plug-and-Play）功能；PCI 独立于处理器的结构形成一种独特的中间缓冲器设计，将中央处理器子系统与外围设备分开。用户可随意增设多种外围设备。

PCI 总线信号分为地址线、数据线、接口控制线、仲裁线、系统线、中断请求线、高速缓存支持和出错报告信号线等，共 188 根。

系统信号线有时钟信号线 CLK 和复位信号线 $\overline{\text{RST}}$。CLK 信号是 PCI 总线上所有设备的一个输入信号，为 PCI 总线上所有设备的 I/O 操作提供同步定时。$\overline{\text{RST}}$ 使各信号线的初始状态处于系统规定的初始状态或高阻态。

地址/数据总线 AD_{31}～AD_0 是分时复用的信号线。

$\overline{C/BE_3}$ ～ $\overline{C/BE_0}$ 称为“命令/字节使能”信号，也是复用线。在传输数据阶段，它们指明所传数据的各字节通路；在传送地址阶段，这 4 条线决定了总线操作的类型，包括 I/O 读、I/O 写、存储器读、存储器写、存储器多重写、中断响应、配置读、配置写和双地址周期等。为了实现即插即用功能，PCI 部件内都置有配置寄存器，配置读和配置写命令就是用于系统初始化时，对这些寄存器进行读/写操作。

PAR 信号为校验信号，用于对 AD_{31}～AD_0 和 $\overline{C/BE_3}$ ～ $\overline{C/BE_0}$ 的偶校验。

接口控制信号有成帧信号 $\overline{\text{FRAME}}$、目标设备就绪信号 $\overline{\text{TDRY}}$、始发设备就绪信号 $\overline{\text{IRDY}}$、停止传输信号 $\overline{\text{STOP}}$、初始化设备选择信号 IDSEL、资源封锁信号 $\overline{\text{LOCK}}$ 和设备选择信号 $\overline{\text{DEVSEL}}$。PCI 总线采用独立请求仲裁方式，每个 PCI 始发设备都有一对总线仲裁线 REQ# 和 $\overline{\text{GNT}}$ 直接连到 PCI 总线仲裁器，各始发设备使用总线时，分别独立地向 PCI 总线仲裁器发出总线请求信号 $\overline{\text{REQ}}$，由总线仲裁器根据系统规定的判决规则决定把总线使用权赋给某设备。

除此以外，还有中断申请、电源、地及一些其他信号线。

3．PCI Express 总线

2001 年春季，Intel 公司提出了用新技术取代 PCI 总线和多种芯片的内部连接，并称为第三代 I/O 总线技术。2001 年年底，包括 Intel、AMD、DELL 和 IBM 在内的 20 多家公司开始起草新技术规范，并在 2002 年完成，将其正式命名为 PCI Express。PCI Express 采用目前业

电子工业出版社

内流行的点对点串行连接，比起 PCI 以及早期的计算机总线的共享并行架构，每个设备都有自己的专用连接，而且可以显著提高数据传输速率。

PCI Express 接口标准根据总线位宽不同而有所差异，包括 X1、X4、X8 和 X16 共 4 种。它支持热拔插，支持的 3 种电压分别为+3.3 V、3.3 V_{aux} 和+12 V。PCI Express 可以从 1 条通道连接到 32 条通道，有非常强的伸缩性，可满足不同系统设备对数据传输带宽的需求。

PCI Express 的连接建立在一个双向序列的点对点连接基础之上，称为“传输通道”。PCI Express 总线的传输依赖于多层协议，由物理层、数据链接层和交换层构成。

① 物理层：PCI Express 采用串行方式传输数据，每组点对点数据传输通道使用两个单向的低电压差分信号（LVDS）进行传输。多传输通道上的数据传输采取交叉存取，这意味着，连续字节交叉存取在不同的通道上，这一特性被称为“数据条纹”，需要复杂的硬件支持连续数据的同步存取，也对链接的数据吞吐量要求极高。与其他高速数据传输协议一样，时钟信息必须嵌入信号中。在物理层，PCI Express 采用常见的 8B/10B 代码方式来确保连续的 1 和 0 字符串长度符合标准，保证接收端不会误读。

② 数据链接层：数据链接层采用按序的交换层数据包（Transaction Layer Packet，TLP），按 32 位循环冗余校验码进行数据保护，采用检错重传机制（Ack and Nak signaling）保证传输的正确性，无应答或等待超时的数据包会被重新传输。这些内容存储在数据链接层的缓存内，以确保数据包的传输不受噪声干扰。Ack 和 Nak 应答信号由数据链接层传送，这些包被称为数据链接层信息包（Data Link Layer Packet，DLLP）。DLLP 也传送两个互连设备的交换层之间的流控制信息并实现电源管理功能。

③ 交换层：PCI Express 采用分离交换（数据提交和应答在时间上分离），以保证传输通道在目标端设备等待发送回应信息时传送其他数据信息。为保证传输质量，交换层使用可信性流控制传输模式。发送端会在发送数据时统计每一次发送数据包所占用的可信信号量，直至达到接收端初始可信信号量的最高值。接收端在处理完毕缓存中的数据包后，会回送给发送端一个比初始值更大的可信信号量，通知对方数据已正确传输。

8.2 工作站

8.2.1 配置和功能

工作站是由计算机、相应的外部设备以及成套的应用软件包所组成的信息处理系统，能够完成用户交给的特定任务，在编程、计算、文件书写、存档以及通信等方面给专业工作者以综合的帮助。常见的工作站有计算机辅助设计（Computer aided design，CAD）工作站（或工程工作站），办公自动化（office automation，OA）工作站和图形工作站等，不同任务的工作站有不同的硬件/软件配置及功能，具体如表 8-2 所示。

8.2.2 分类

工作站根据操作系统的不同，一般分为基于 UNIX 系统的工作站和基于 Windows 系统的工作站。基于 UNIX 系统的工作站多采用 RISC 处理器、针对特定场景优化的软硬件系统，彼此互不兼容，Linux 是一个类 UNIX 系统，是一个开源免费的系统，以 Linux 为架构的工作

電子工業出版社

表 8-2　工作站软、硬件配置及功能

工作站	硬件配置	软件配置	功　能
CAD 工作站	计算机、带有功能键的 CRT 终端、光笔、平面绘图仪、数字化仪、打印机	操作系统、编译程序、数据库和数据库管理系统、2D/3D 绘图软件，计算/分析软件包	完成各种机械/电气设计任务
OA 工作站	计算机、办公用终端设备（如电传打字机、交互式终端、传真机、激光打印机、智能复印机）、通信设施（如局部区域网、程控交换机、公用数据网、综合业务数字网）	操作系统、编译程序、服务程序、数据库和数据库管理系统、文字/表格处理软件、编辑软件、专门业务活动的软件包（如人事管理、财务管理、行政事务管理等）	完成各种办公信息处理
图形工作站	顶级计算机、超强性能的显卡、图像数字化设备（包括电子/光学/机电扫描设备/数字化仪）、图像输出设备、交互式图像终端	操作系统、编译程序、图像处理软件包	完成各种图像处理任务

站能最大程度的降低拥有成本，可以免费使用 Linux 系统及基于 Linux 系统的开源软件。基于 Windows 系统的工作站多采用英特尔至强处理器，使用稳定的 Windows 7 32/64 位操作系统，采用符合专业图形标准（OpenGL4.4 和 DirectX 12）的图形系统，再加上高性能的存储、I/O 接口、网络等子系统，来满足专业软件运行的要求。

此外，根据体积和便携性，工作站还可分为台式工作站和移动工作站。台式工作站类似普通台式计算机，体积较大，没有便携性可言，但性能强劲，适合专业用户使用。移动工作站其性能相当于一台高性能的笔记本电脑，但在系统稳定性和专业计算能力方面比普通笔记本电脑有较大程度提升。

8.2.3　工作站的特点

1．CPU

一般，传统的工作站 CPU 并非 Intel 和 AMD 公司 CPU，而是使用 RISC 架构处理器，如 PowerPC 处理器、SPARC 处理器、Alpha 处理器等，相应的操作系统一般为 UNIX 或者其他专门的操作系统。20 世纪 80 年代早期，RISC 架构的处理器在单位成本上提供比 CISC 处理器高几个数量级的性能。也正是在这个时期，CISC 处理器的一族（Intel 的 x86 系列）渐渐侵占市场份额。到了 90 年代中期，Intel 的 CPU 终于拥有了同 RISC 相提并论的性能，并将后者赶进了一个狭窄的市场。

目前，英特尔至强系列处理器是专为服务器和工作站来设计的专用 CPU，具有强大的处理性能。其突出特点是集成了 QPI 内存控制器，内存和 CPU 之间的数据通信不再经过北桥控制芯片，突破了 CPU 和内存之前数据交换的瓶颈。随着 Intel 公司 14nm 技术的普及，Intel 至强四核或者六核以其强大的运算能力成为当前服务器/工作站 CPU 的不二选择。

2．显卡

作为图形工作站的主要组成部分，一块性能强劲的 3D 专业显卡的重要性从某种意义上来说甚至超过了处理器。与针对游戏、娱乐市场为主的消费类显卡相比，3D 专业显卡主要面对的是三维动画（如 3ds MAX、Maya、Softimage|3D）、渲染（如 LightScape、3ds VIZ）、CAD（如 AutoCAD、Pro/Engineer、Unigraphics、SolidWorks）、模型设计（如 Rhino）以及部分科学应用等专业 OpenGL 应用市场。图形工作站的硬件无论是速度、稳定性还是软件的兼容性都很重要。目前，整个 3D 专业显卡市场已经成了 NVIDIA 和 AMD 收购的 ATI 的天下。由于工作需求不同，工作站的显卡与消费类显卡差别比较大。不过，目前 NVIDIA 和 ATI 的趋势是将娱乐级产品和专业级产品统一到几乎完全相同的硬件架构下，甚至是完全相同的芯片。

電子工業出版社

当前主流显卡有 NVIDIA 的 GeForce 系列、Quadro 系列、TITAN 系列以及 AMD 的 Radeon 系列。

3．内存

目前，主流工作站的内存为 ECC（Error Checking and Correcting）内存和 REG（Register）ECC 内存。ECC 主要用在中低端工作站上，通过在原来的数据位上额外增加数据位来实现错误校验和纠错功能。若工作正常时，用户不会发觉数据出过错，只有经过内存的纠错后，计算机的操作指令才可以继续执行；在纠错时，系统的性能有着明显降低，不过这种纠错对服务器等应用而言是十分重要的。高端的工作站和服务器上用的都是 REG ECC 内存，REG ECC 内存一定是 ECC 内存，而且多加了个寄存器缓存，数据存取速度大大加快。

4．硬盘

目前用于工作站系统的硬盘主要有固态硬盘、机械硬盘和混合硬盘，按照硬盘接口种类，又可以分为 SAS 硬盘、SATA（Serial ATA）硬盘、SCSI 硬盘。工作站对硬盘的要求介于普通台式机和服务器之间，因此低端的工作站一般使用与台式机一样的 SATA 或者 SAS 硬盘。中高端的工作站会使用 SAS 或固态硬盘。

磁盘冗余阵列 RAID（Redundant Array Of Independent Disk, Redundant Array Of Inexpensive Disk）对于 SATA 硬盘和 SAS 硬盘管理必不可少，多个 SATA 硬盘按照不同的 RAID 模式组建存储阵列。常用的有 RAID 0 1 5 6 10，除了 RAID 0，其他模式必须要 3 块或者 3 块以上的硬盘。磁盘阵列的好处有两个：成倍增加磁盘存取速度，以及保证数据的安全性，一块或者多块磁盘坏掉，更换磁盘后可以通过 RAID 重构。

5．高性能的 3D 图形配置

工作站在很大程度上与相图形软件结合。当前的图形软件上已经发展至 OPENGL 4.4 和 Direct 12，只支持之前版本的专业图形卡以及非专业的游戏显卡已经逐渐无法满足。同时，传统的 CRT 和普通液晶已经越来越不能满足专业图形处理和模拟仿真等在色彩和屏分的需求，高端专业的显示器已经成为主流。

8.3 服务器

服务器也称为伺服器，可提供计算服务给各终端。由于服务器需要响应终端的服务请求，因此服务器应具备承担服务并且保障服务的能力。

8.3.1 分类

按照体系架构来分，服务器主要分为两类：非 x86 服务器和 x86 服务器。

1．非 x86 服务器

非 x86 服务器包括大型机、小型机和 UNIX 服务器，使用 RISC 或 EPIC（并行指令代码）处理器，并且主要采用 UNIX 和其他专用操作系统。RISC 处理器主要有 IBM 的 Power 和 PowerPC 处理器，SUN 与富士通公司合作研发的 SPARC 处理器；EPIC 处理器主要是 Intel 研发的安腾处理器等。这种服务器价格昂贵、体系封闭，但是稳定性好、性能强，主要用在

電子工業出版社

金融、电信等大型企业的核心系统中。

2．x86 服务器

x86 服务器又称 CISC 架构服务器，即通常所讲的 PC 服务器，基于 PC 体系结构，使用 Intel 或其他兼容 x86 指令集的处理器芯片和 Windows 操作系统，价格便宜、兼容性好、稳定性较差、安全性不算太高，主要用在中小企业和非关键业务中。

服务器也可按应用层次来划分，根据服务器在网络中应用的层次（或服务器的档次来）可分为：入门级服务器、工作组级服务器、部门级服务器和企业级服务器。随着应用层次的提高，各级服务器的特性依次提高，具体如表 8-3 所示。

表 8-3　各级服务器特性

服务器	特　　性
入门级服务器	有些基本硬件的冗余，如硬盘、电源、风扇等，但不是必须 通常采用 SCSI 接口硬盘，也有采用 SATA 串行接口 部分部件支持热插拔，如硬盘和内存等，但不是必须 通常只有一个 CPU，但不是绝对 主要采用 Windows 或者 NetWare 网络操作系统 所连终端比较有限，通常为 20 台左右 可以充分满足办公室中小型网络用户的文件共享、数据处理、Internet 接入及简单数据库应用的需求；在稳定性、可扩展性以及容错冗余等方面性能较差，仅适合无大型数据库数据交换，日常工作网络流量不大，不需长期不间断开机的小型企业。
工作组服务器	通常仅支持单或双 CPU 结构（但不是绝对的，特别是 SUN 的工作组服务器就有能支持多达 4 个处理器的工作组服务器） 支持大容量的 ECC 内存和增强服务器管理功能的 SM 总线 采用 Intel 服务器 CPU 和 Windows/NetWare 网络操作系统，有的采用 UNIX 操作系统 只能连接一个工作组，50 台左右终端 较入门级服务器的性能有所提高，但容错和冗余性能仍不完善，不能满足大型数据库系统的应用
部门级服务器	一般支持双 CPU 以上的对称处理器结构，具备比较完全的硬件配置，如磁盘阵列、存储托架等 集成了大量的监测及管理电路，具有全面的服务器管理能力，可监测如温度、电压、风扇、机箱等状态参数 具有优良的系统扩展性，能够满足用户在业务量迅速增大时及时在线升级系统 CPU 一般采用 IBM、SUN 和 HP 各自开发的 RISC 结构芯片，一般采用 UNIX 或 Linux 操作系统 可连接 100 个左右终端，其硬件配置相对较高 需要安装比较多的部件，所以机箱通常较大，采用机柜式
企业级服务器	最起码采用 4 个以上 CPU 的对称处理器结构，有的高达几十个 具有独立的双 PCI 通道和内存扩展板设计，具有高内存带宽，超强的数据处理能力和群集性能等 硬盘、电源、RAM、PCI 和 CPU 等具有热插拔性能 具有高度的容错能力、故障预报警功能、在线诊断 机箱更大，一般为机柜式的，有的还由几个机柜来组成 配置固态硬盘，让高速存储更加的接近处理器，并将共享存储网络这个潜在的瓶颈剔除掉 适合运行在需要处理大量数据、高处理速度和对可靠性要求极高的金融、证券、交通、邮电、通信或大型企业

8.3.2　硬件特点

1．结构

服务器系统的硬件构成与 PC 有众多相似之处，主要的硬件构成仍然包含如下几个主要部分：中央处理器、内存、芯片组、I/O 总线、I/O 设备、电源和机箱等，但服务器具备比 PC 更高的硬件性能；同时，对数据相当敏感的应用还要求服务器提供数据备份功能。

2．CPU

服务器的 CPU 仍然按照其指令系统来区分，通常分为 CISC 和 RISC 两类，后来又出现了一种 64 位的超长指令集 VLIM（Very Long Instruction Word）架构的 CPU。

电子工业出版社

CISC 处理器是指英特尔生产的 x86 系列 CPU 及其兼容 CPU（其他厂商如 AMD，VIA 等生产的 CPU），基于 PC 体系结构，一般都是 32 位结构，所以也被称为 IA-32 CPU（IA，Intel Architecture，Intel 架构）。

RISC 处理器相对于 CISC 处理器，不仅精简了指令系统，还采用了“超标量和超流水线结构”架构，在同等频率下，采用 RISC 架构的 CPU 比 CISC 架构的 CPU 性能高很多，这是由 CPU 的技术特征决定的。

8.3.3 外形

1．机架式

信息服务企业选择服务器时应先考虑服务器的体积、功耗和发热量等物理参数，因为信息服务企业通常使用大型专用机房统一部署和管理大量的服务器资源，机房通常设有严密的保安措施、良好的冷却系统、多重备份的供电系统，其造价相当昂贵。如何在有限的空间内部署更多的服务器直接关系到企业的服务成本，通常选用机械尺寸符合 19 英寸工业标准的机架式服务器。

机架式服务器的外形看来不像计算机，而像交换机，有 1U（1U=1.75 英寸≈4.445 cm）、2U、4U 和 8U 等规格，可安装在标准的 19 英寸机柜里面。通常 1U 的机架式服务器最节省空间，但性能和可扩展性较差，适合一些业务相对固定的使用领域；4U 以上的产品性能较高，可扩展性好，一般支持 4 个以上的高性能处理器和大量的标准热插拔部件，管理十分方便，适合大访问量的关键应用，但体积较大。

2．刀片式

刀片式服务器是指在标准高度的机架式机箱内可插装多个卡式的服务器单元，实现高可用和高密度。每块“刀片”实际上是一块系统主板，可以通过“板载”硬盘启动自己的操作系统，如 Windows Server 2016、Linux 等，类似一个个独立的服务器。在这种模式下，每块主板运行自己的系统，服务于指定的不同用户群，相互之间没有关联，因此相较于机架式服务器和机柜式服务器，单片主板的性能较低。不过，管理员可以使用系统软件将这些主板集合成一个服务器集群。在集群模式下，所有主板可以连接起来提供高速的网络环境，并同时共享资源，为相同的用户群服务。在集群中插入新的“刀片”，可以提高整体性能。每块“刀片”都是热插拔的，所以系统可以轻松地进行替换，并且将维护时间减少到最小。

3．塔式

塔式服务器的外形以及结构都与平时使用的台式计算机差不多。当然，服务器的主板扩展性较强、插槽也多，所以个头比普通主板大一些。塔式服务器的主机机箱比标准的 ATX 机箱要大，一般会预留足够的内部空间，以便日后进行硬盘和电源的冗余扩展。

塔式服务器的机箱比较大，相应配置也很高，冗余扩展更齐备，所以应用范围非常广，应该是使用率最高的一种服务器。我们平时常说的通用服务器一般是指塔式服务器。

4．机柜式

在一些高档企业服务器中，由于内部结构复杂，内部设备较多，有时会将许多不同的设备单元或几个服务器都放在一个机柜中，这种服务器就是机柜式服务器。机柜式通常由机架

電子工業出版社

式服务器、刀片式服务器再加上其他设备组合而成。

8.3.4 计算机、工作站和服务器

1. 服务器和计算机

服务器与计算机的不同点很多：计算机在一个时刻通常只为一个用户服务，而服务器可同时为多个终端服务；计算机是通过终端给用户使用的，而服务器是通过网络给客户端用户使用的。此外，服务器需要连续的工作在 24 小时环境，是一种配置更高的计算机，需要更可靠的硬件支持，管理资源并为用户提供服务。

从硬件层次上来说，服务器与一般计算机一样，均由 CPU、内存、主板、显卡、硬盘等组成，由于服务器和普通计算机相比需要具备更强的处理数据能力，因此在主板上通常会安装多个处理器、内存、硬盘。从操作系统层次上来说，一般计算机使用的是 Windows XP/7/10 等系统，服务器一般使用 Windows Server 2008/2012/2016 和 Linux、UNIX 等服务器系统。

2. 服务器和工作站

虽然工作站与服务器一样，都是一台高性能的计算机，但二者侧重不同，与服务器强调稳定性不同，工作站侧重工作时的高效性，具有极强的信息、图形和图像处理能力。此外，连接到服务器的终端机通常为工作站。

最后，借用网上看到的一句话，计算机、工作站和服务器分别代表的是个人需求（基础）、专业需求（图像）、大众需求（后台）。三者互相联系，互相区别。

8.4 SoC 与嵌入式系统

8.4.1 SoC

从狭义角度讲，片上系统（System on Chip，SoC）是信息系统核心的芯片集成，是将系统关键部件集成在一块芯片上；广义而言，它是一个微小型系统，是在单片上集成系统级、多元化的大规模功能模块，从而构成一个能够处理各种信息的集成系统。SoC 的构成可以是系统级芯片控制逻辑模块、微处理器/微控制器 CPU 内核模块、数字信号处理器 DSP 模块、嵌入的存储器模块、与外部进行通信的接口模块、含有 ADC/DAC 的模拟前端模块、电源提供和功耗管理模块；无线 SoC 则还有射频前端模块、用户定义逻辑（可由 FPGA 或 ASIC 实现）以及微电子机械模块；更重要的是，SoC 芯片内嵌有基本软件模块或可载入的用户软件等。SoC 通常是客户定制的，或是面向特定用途的标准产品。

SoC 设计中有如下几个关键技术。

1. IP 核复用技术

IP 核字面上的意思是知识产权核或知识产权模块，实际是指将一些在数字电路中常用，但比较复杂的功能块，如 FIR 滤波器、SDRAM 控制器、PCI 接口等设计成可修改参数的模块，用于 SoC 或复杂的 ASIC 设计中。随着 CPLD/FPGA 的规模越来越大，设计越来越复杂，设计者的主要任务是在规定的时间周期内完成复杂的设计。IP 核复用技术能避免重复劳动，大大减轻工程师的负担，因此这是一个发展趋势，将大大缩短产品上市时间。

電子工業出版社

IP 核主要分为软核、固核和硬核三种。这种分类主要依据产品交付的方式，而这三种 IP 核实现方法也各具特色，具体区别如表 8-4 所示。

表 8-4　IP 核分类及特点

IP 核	提交形式	用户设计开发成本	灵活性	可靠性	价　格
软核	RTL 级代码	高，设计周期长	较大，易于移植，	较差，用户可以修改配置	便宜
固核	门级网表	中	中	中	中
硬核	物理版图文件	低，设计周期短	较差，不易移植	较强，易于保护	高

2．软硬件协同设计

受电子技术特别是可编程技术的限制，传统的设计方法是将硬件和软件分为两个独立的部分进行设计。在整个设计过程中，通常采用硬件优先的原则，一般先进行硬件设计，再在硬件设计平台上进行软件设计。随着电子技术的发展，各种大规模可编程集成电路得到广泛的应用，传统的设计方法的局限性已成为限制可编程芯片充分发挥性能的障碍，在此基础上，人们开始研究软硬件协同设计技术。

软硬件协同设计是依据系统设计为目标，通过综合分析系统软硬件功能及现有资源，对系统中的软件和硬件部分使用统一的描述和工具进行集成开发，可完成全系统的设计验证并跨越软硬件界面进行系统优化。软硬件协同设计是使软件设计和硬件设计作为一个有机的整体进行并行设计，实现软件与硬件的最佳结合，从而使系统获得高效工作能力。软硬件协同设计的基本思路如图 8-6 所示。

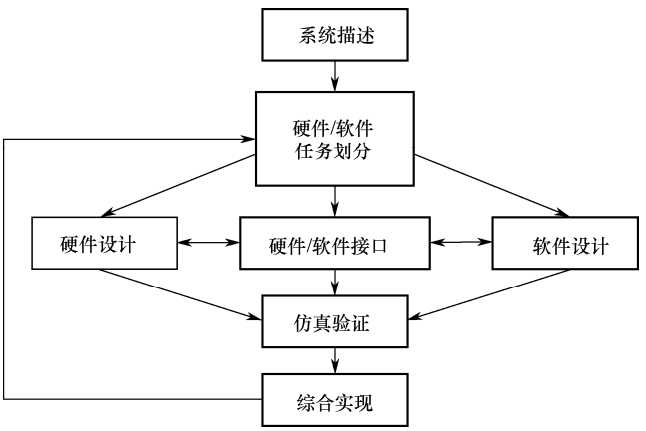


图 8-6　软硬件协同设计流程

可以看出，软硬件协同设计最主要的优点是在设计过程中，硬件和软件设计是相互作用的，这种相互性体现在设计过程的各阶段和各层次，充分实现了软硬件的协同性。在软硬件任务划分时就考虑了现有软件和硬件的资源，在软硬件功能设计和仿真评价过程中，软件和硬件是互相支持的。这就使得软硬件功能模块能够在设计开发的早期互相结合，从而及早发现和解决系统设计的问题，避免了在设计开发后期反复修改所带来的一系列问题，有利于充分挖掘系统潜能、缩小体积、降低成本、提高整体效能。

3．SOPC

Altera 公司在 2000 年提出的 SoPC（System on a Programmable Chip）是 SoC 技术与可编程逻辑技术结合的产物，即基于大规模 FPGA 的单芯片系统。Altera 公司同时推出了相应的开发软件 SOPC Builder 和可供使用的 IP 核，还有很多公司开发的 IP 核可供选择。Altera 公

電子工業出版社

司提供了 SoPC 开发的整体方案，所以开发效率高、成本低，且不存在兼容性的问题。

在 Altera 公司所提供的 SoPC 系统中，嵌入式微处理器的 IP 核分软核和硬核两种：硬核可以是 ARM 或其他的微处理器 IP 核；目前最具代表性嵌入式微处理器软核则是 Altera 公司的 Nios 系列软核，以及 Xilinx 公司的 MicroBlaze 核。

8.4.2 嵌入式系统

嵌入式系统是 SoC 的基本结构。在使用 SoC 技术设计的应用电子系统中，可以十分方便地实现嵌入式结构，只要根据系统需要选择相应的内核，再根据设计要求选择之相配合的 IP 模块，就可以完成整个系统硬件结构。SoC 的这种嵌入式结构可以大大缩短应用系统设计开发周期。

从广义上来讲，嵌入式系统是指以应用为中心，以计算机技术为基础，软硬件可裁剪，适应应用系统对功能、可靠性、成本、体积、功耗等严格要求的专用计算机系统；从狭义上理解，嵌入式系统就是嵌入到对象体中的专用计算机系统。嵌入式系统在应用数量上远远超过了各种通用计算机，一台通用计算机的外部设备中就包含了若干嵌入式处理器，键盘、鼠标、显卡、显示器、网卡、声卡和打印机等均是由嵌入式处理器控制的。

1．嵌入式系统硬件及软件组成

嵌入式系统的硬件主要包括嵌入式处理器、嵌入式系统存储器、I/O 接口及常用的 I/O 设备、典型 ARM 处理芯片以及嵌入式互连通信接口。

嵌入式系统软件组成如图 8-7 所示。对于简单的应用，如不使用操作系统或仅使用小型操作系统的嵌入式系统，软件组成也不尽相同。板级支持包（Board Support Package，BSP）和硬件抽象层（Hardware Abstract Layer，HAL）与 PC 的基本输入/输出系统（Basic Input Output System，BIOS）相似。不同的嵌入式微处理器、不同的硬件平台或不同的操作系统，BSP/HAL 也不同。

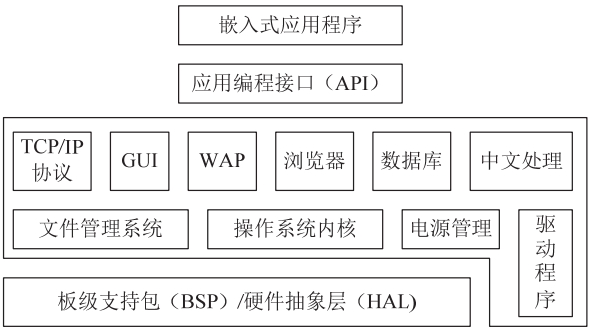


图 8-7　嵌入式系统软件组成

2．嵌入式处理器

嵌入式系统硬件部分的核心是嵌入式处理器。按照嵌入式处理器的应用领域，嵌入式处理器从广义上可以分为 4 类：微控制器（Micro Controller Unit，MCU）、微处理器（Micro Processor Unit，MPC）、数字信号处理器（Digital Signal Processor，DSP）、片上系统。

① 嵌入式微控制器：典型代表是单片机，将整个计算机系统集成到一块芯片中。嵌入式微控制器一般以某种微处理器内核为核心，根据典型应用，在芯片内部集成了 ROM/EPROM、

電子工業出版社

RAM、总线、定时/计数器、看门狗、I/O、串行口、A/D 及 D/A 等必要的功能部件和外设。微控制器是目前嵌入式系统的主流，其片上外设资源比较丰富，适合工业控制场合，比较典型的产品有 8051、P51XA、MCS-251、MCS-96/196/296、MC68HC05/11/12/16 以及 68300 等。

② 嵌入式微处理器：微处理器是由通用计算机中的 CPU 演变而来的。在应用中，将微处理器装配在专门设计的电路板上，只保留和嵌入式应用有关的功能，这样可以大幅度减小系统体积和功耗。嵌入式微处理器具有体积小、重量轻、成本低和可靠性高等优点，目前的主流产品有 PowerPC、MIPS 和 ARM 系列等。

③ 嵌入式数字信号处理器：嵌入式 DSP 有两个发展来源，一是 DSP 经过单片化、电磁兼容性改造，增加片上外设成为嵌入式 DSP，如 TI 的 TMS320C2000/C5000 等；二是在通用单片机或 SoC 中增加 DSP 协处理器，如 Infineon 的 Tricore。

④ 嵌入式片上系统：嵌入式 SoC 可分为通用和专用两类。通用系列包括 Infineon 的 Tricore，STMicroelectronics 的 STM32 系列等。专用 SoC 一般专用于某个或某类系统中，不为一般用户所知，代表性的产品是 Philips 的 Smart XA，它是将 XA 单片机内核和支持超过 2048 位复杂 RSA 算法的 CPU 单元制作在一块硅片上，形成一个可加载 JAVA 和 C 语言的专用 SoC，可用于公众互联网的安全方面。

3．嵌入式操作系统

嵌入式操作系统是一种用途广泛的系统软件，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。嵌入式操作系统负责嵌入式系统的全部软硬件资源的分配、任务调度、控制、协调并发活动。嵌入式操作系统可充分体现其所在系统的特征，并通过装卸某些模块来达到系统所要求的功能。

嵌入式操作系统的主要特点有实时性、可移植性、内核小型化和可裁剪等。目前，在嵌入式领域广泛使用的操作系统有嵌入式实时操作系统 μC/OS、嵌入式 Linux、Windows CE、Android 和 VxWorks 等。

① μC/OS 是源码公开的实时嵌入式操作系统，提供了嵌入式系统的基本功能，核心代码短小精悍，但对于大型商用嵌入式系统而言还是有些简单。目前已被移植到很多微处理器上运行，如 ARM 系列，Intel 公司的 8051、80x86 系统，Motorola 公司的 PowerPC 等系列。

② Linux 也是一套免费的、开源代码，适用于多用户嵌入式系统。Linux 存在着不同版本，但都使用了 Linux 内核，可以安装在各种计算机硬件设备中，如手机、平板电脑、路由器、视频游戏控制台等。

③ Windows CE 操作系统是微软公司于 1996 年发布的一种嵌入式操作系统，在 PDA、Pocket PC、Smart Phone（智能手机）、工业控制和医疗设备方面用得较多。它是一款简洁、高效的多平台操作系统，是从整体上为有限资源的平台设计的多线程、完全优先级、多任务的操作系统。近年来，微软公司推出了针对移动设备应用的 Windows Mobile 操作系统，主要面向个人移动电子消费市场。

④ Android 是由 Google 公司推出的以 Linux Kernel 为核心的移动操作系统，也是开源代码，用户能够按照需要对其进行裁剪或扩展。Android 不仅是全球最受欢迎的智能手机操作系统，同时在平板电脑市场占有率也逐年上升。采用该系统的主要有国内众多的手机厂商和嵌入式教学实验平台。

⑤ VxWorks 是美国 Wind River System 公司于 1983 年设计开发的一个实时操作系统。它

电子工业出版社

既是一个操作系统，又是一个可以运行的最小基本程序；有 BSP，可以减小驱动程序的编写过程；具有强大的调试能力，可以在没有仿真器的情况下，通过串口调试，并具有丰富的函数库。

8.5 多核处理器

多核处理器是指在一枚处理器中集成两个或多个完整的计算引擎（内核），此时处理器能支持系统总线上的多个处理器，由总线控制器提供所有总线控制信号和命令信号。

多核处理器主要分原生多核和封装多核。原生多核指的是真正意义上的多核，最早由 AMD 提出，每个内核之间都是完全独立的，都拥有自己的前端总线，不会造成冲突，即使在高负载状况下，每个内核都能保证自己的性能不受太大的影响，通俗地说，原生多核的抗压能力强，但是需要先进的工艺，每扩展一个内核都需要很多的研发时间。封装多核是只把多个内核直接封装在一起，如 Intel 早期的 PD 双核系列就是把两个单核直接封装在一起，但两个内核只能共同拥有一条前端总线，在两个内核满载时，它们会争抢前端总线，导致性能大幅度下降。与原生多核相比，封装多核性能差了很多，而且后者成本比较高，但是封装多核的发展要比原生快得多。

8.5.1 发展历程

多核技术是处理器发展的必然产物。推动微处理器性能不断提高的因素主要有两个：半导体工艺技术的飞速进步和体系结构的不断发展。这两个因素是相互影响，相互促进的：半导体工艺技术的每一次进步都为微处理器体系结构的研究提出了新的问题，开辟了新的领域；体系结构的进展又在半导体工艺技术发展的基础上进一步提高了微处理器的性能。一般说来，工艺技术的发展使得处理器性能提高约 20 倍，体系结构的发展使得处理器性能提高约 4 倍，编译技术的发展使得处理器性能提高约 1.4 倍。自第一款通用型微处理器 4004 诞生以来，处理器芯片一直遵循着“摩尔定律”的规律发展。但是到了 2005 年，这种规律性的东西却很难维持。当处理器芯片主频接近 4 GHz 时，Intel 和 AMD 发现，速度也会遇到极限，单纯的主频提升，已经无法明显提升系统整体性能。此外，随着功率增大，散热问题越来越成为一个无法逾越的障碍。当晶体管数量增加导致功耗增长超过性能增长速度后，处理器的可靠性就会受到致命性的影响，于是多核处理器解决方案应运而生。

20 世纪 90 年代末，CMP（单芯片多处理器）技术已出现，用来替代复杂性较高的单线程 CPU。IBM、HP、Sun 等高端服务器厂商相继推出了多核服务器 CPU。2005 年 4 月，Intel 仓促推出简单封装双核的 Pentium D 和 Pentium 4 至尊版 840。AMD 在之后也发布了双核皓龙（Opteron）和速龙（Athlon）64 位 X2 处理器。但真正的“双核处理器”被认为是 2006 年 Intel 基于酷睿（Core）架构的处理器。2006 年 11 月，Intel 又推出面向服务器、工作站和高端个人计算机的至强（Xeon）5300、酷睿双核和四核至尊版系列处理器。与上一代台式机处理器相比，酷睿双核处理器在性能方面提高 40%，功耗反而降低 40%。由于功耗已成为用户在性能之外所考虑的首要因素，两大处理器巨头都在宣传多核处理器时，强调其“节能”效果。Intel 发布了功耗仅为 50 W 的低电压版四核至强处理器。AMD 的“Barcelona”四核处理器的功耗没有超过 95 W。Intel 于 2008 年推出了 Core i7 处理器，它是 64 位四核心 CPU，

電子工業出版社

沿用 x86-64 指令集，并以 Intel Nehalem 微架构为基础，取代 Intel Core 2 系列处理器。

Intel 的多核芯片采用“横向扩展”（而非“纵向扩充”）方法提高性能，实现了“分治法”战略：通过划分任务，线程应用能够充分利用多个执行内核，并可在特定的时间内执行更多任务。操作系统会利用所有相关的资源，将每个执行内核作为分立的逻辑处理器。通过在两个执行内核之间划分任务，多核处理器可在特定的时钟周期内执行更多任务。

另一方面，随着向多核处理器的移植，现有软件不需被修改就可支持多核平台。为了充分利用多核技术，应用开发人员需要在程序设计中融入更多思路，但设计流程与对称多处理系统的设计流程相同，并且现有的单线程应用也将继续运行。

8.5.2 多核技术

1．技术种类

多核处理器体系结构可分为单芯片多处理器（CMP）与同时多线程处理器（Simultaneous Multithreading，SMT），这两种体系结构可以充分利用一些应用的指令级并行性和线程级并行性，从而显著提高了应用的性能。CMP 是在一个芯片上集成多个微处理器内核，每个内核实质上都是一个单线程或者比较简单的多线程微处理器，多个内核可以并行地执行程序代码，因而具有了较高的线程级并行性。CMP 还能充分利用不同应用的指令级并行和线程级并行。

从体系结构的角度看，SMT 比 CMP 对处理器资源利用率要高，在克服线延迟影响方面更具优势。CMP 相对 SMT 的最大优势在于其模块化设计的简洁性，复制简单设计非常容易，指令调度更加简单。此外，SMT 中多个线程对共享资源的争用也会影响其性能，而 CMP 对共享资源的争用要少得多。因此当应用的线程级并行性较高时，CMP 性能一般要优于 SMT。在设计上，更短的芯片连线使 CMP 比长导线集中式设计的 SMT 更容易提高芯片的运行频率，从而在一定程度上起到性能优化的效果。

2．技术原理

① 核结构研究：多核处理器的结构分同构和异构两类，同构是指内部核的结构是相同的，而异构反之。此外，核所用的指令系统对多核处理器性能很重要，多核之间采用相同的指令系统还是不同的指令系统，能否运行操作系统等，也是研究的内容之一。

② 多级 Cache 设计与一致性问题：处理器与主存间的速度差距对 CMP 来说是个突出的矛盾，因此必须使用多级 Cache 来缓解，目前有共享一级 Cache、共享二级 Cache 以及共享主存的 CMP。Cache 自身的体系结构设计也直接关系到系统整体性能，在 CMP 结构中共享 Cache 或独有 Cache、在一块芯片上建立几级 Cache 等研究，对整个芯片的尺寸、功耗、布局、性能以及运行效率等都有很大的影响。

另一方面，多级 Cache 引发了一致性问题。在处理器系统结构中广泛采用的 Cache 一致性模型有顺序一致性模型、弱一致性模型和释放一致性模型等。与之相关的 Cache 一致性机制主要有基于总线的侦听协议和基于目录的目录协议，目前 CMP 系统大多采用前者。

③ 核间通信技术：CMP 处理器的各内核执行的程序之间有时需要进行数据共享与同步，因此其硬件结构必须支持核间通信。目前比较主流的片上高效通信机制有两种，一种是基于总线共享的 Cache 结构，一种是基于片上的互连结构。

总线共享 Cache 结构是指每个 CPU 内核拥有共享的二级或三级 Cache，用于保存比较常

電子工業出版社

用的数据，并通过连接核心的总线进行通信。这种系统的优点是结构简单，通信速度高，缺点是基于总线的结构可扩展性较差。

基于片上互连的结构是指每个 CPU 内核具有独立的处理单元和 Cache，各内核通过交叉开关或片上网络等方式连接在一起，内核间通过消息通信。这种结构的优点是可扩展性好，数据带宽有保证; 缺点是硬件结构复杂，且软件改动较大。

将来这两者的竞争结果可能是互相合作，如在全局范围采用片上网络而局部采用总线方式，来达到性能与复杂性的平衡。

④ 总线设计：传统微处理器中，Cache 不命中或访存事件都会对 CPU 的执行效率产生负面影响，而 BIU 的工作效率会决定此影响的程度。当多个 CPU 核心同时要求访问内存或多个 CPU 核心内私有 Cache 同时出现 Cache 不命中事件时，BIU 对这多个访问请求的仲裁机制以及对外存储访问的转换机制的效率决定了 CMP 系统的整体性能。因此，CMP 处理器研究的重要内容之一是寻找高效的多端口 BIU 结构，将多内核对主存的单字访问转为更为高效的猝发（burst）访问，以及寻找对 CMP 处理器整体效率最佳的一次 Burst 访问字的数量模型以及高效多端口 BIU 访问的仲裁机制。

8.5.3 多核处理器开发应用

多核处理器的出现增加了并行的层次性能使得并行程序的开发比以往更难，无论是编程模型、开发语言还是开发工具，并无有效的并行计算解决方案。

首先，并不是所有的操作系统和应用软件都做好了迎接多核平台的准备，尤其是在数十年来均为单一线程开发应用的 x86 服务器领域。

在过去一段长时间里，x86 系统上软件的性能随着来自 Intel 和 AMD 处理器速度越来越快而不断提高，开发者只需对现有软件程序作轻微改动就可让其性能随着硬件性能的上升而不断提升。不过，多核设计概念的出现迫使软件设计人员不得不解决并行性（将单个任务拆分成多个小块以便分别处理之后再重新组合的能力）问题。因为多核处理器和多路系统在服务器市场已经存在多年（在传统的 UNIX 领域），一些运行在 RISC 架构多核多路系统上的应用程序已经被设计成多线程，以更好地利用系统并行处理能力。但是，在 x86 领域，应用程序开发者多年来仍一直停留在单线程设计上，自动的并行化解决方案基本没有，传统的手工式的并行程序开发方式又难以为普通的程序员所掌握。

Intel 很早就通过超线程技术实现了逻辑上的双核处理器系统，可以并行计算。但这不过是对处理器闲置资源的一种充分利用而已，并且这种利用只有在特定的条件下，尤其是针对流水线比较长且两种运算并不相互交叉的时候才会有较高的效率。IBM 的 Power5 架构也需要与最新的操作系统进行融合，加上运行在其上的软件，才有可能利用并发多线程。

8.6 并行计算与分布式计算

8.6.1 并行计算

并行计算（Parallel Computing）是相对于串行计算来说的，是指同时使用多种计算资源解决计算问题的过程，是提高计算机系统计算速度和处理能力的一种有效手段。它的基本思

電子工業出版社

想是用多个处理器来协同求解同一问题，即将被求解的问题分解成若干部分，各部分均由一个独立的处理机来并行计算。并行计算系统既可以是专门设计的、含有多个处理器的超级计算机，也可以是以某种方式互连的若干独立计算机构成的集群，通过并行计算集群完成数据的处理，再将处理的结果返回给用户。

并行计算可分为时间上的并行和空间上的并行。时间上的并行就是指流水线技术，空间上的并行则是指多个处理机并发的执行计算，即通过网络将两个以上的处理机连接起来，达到同时计算同一个任务的不同部分，或者单个处理机无法解决的大型问题。

1．并行计算机系统分类

并行计算科学中主要研究的是空间上的并行问题，根据计算机系统中数据流和指令流的多倍性，可以分为 4 类：单指令流多数据流（Single Instruction Multiple Data，SIMD）、多指令单数据流（Multiple Instruction Single Data，MISD）和多指令流多数据流（Multiple Instruction Multiple Data，MIMD），我们常用的串行机也叫做单指令流单数据流（Single Instruction Single Data，SISD）。

① SISD 计算机系统：单一处理器执行单一的指令流以实现对保存在单一可访问存储器内的数据操作。SISD 计算机的典型例子是单处理器系统，其中没有并行处理方式。

② SIMD 计算机系统：单一指令可以同步控制多个处理器，每个处理器都有一个相关的数据存储器，所以一条指令可以在不同的数据组上完成相同的操作。

③ MISD 计算机系统：实现对顺序数据进行多个处理器的操作。每个处理器执行不同的指令序列。在实际系统中还没有完全的 MISD 计算机出现。

④ MIMD 计算机系统：多个处理器并行处理完成不同的指令序列，对不同的数据进行加工操作，实际上是多处理器并行系统。在 MIMD 的组织下，各处理器是通用的，每个处理器都能处理所有数据并完成相应数据运算的指令。

MIMD 系统还可以按照各处理器的通信方式进一步进行划分，目前普遍采用的并行多处理器组织方式有对称多处理机（SMP）、机群系统（Cluster）和非均匀存储器（NUMA）三种。在对称多处理机系统中，多个相同或相似的处理器共享一个存储器，每个处理器存取共享存储器中的程序和数据，并经由次存储器相互通信，是通常意义上的多处理器计算机。在 SMP 存储系统中，最重要的问题是存储一致性：每个处理器均具有独立的 Cache，当某一行数据出现在不同处理器的 Cache 中时，如果对一个 Cache 内容进行修改，那么其他 Cache 以及主存中内容也应得到更新。机群系统是由一组完整的计算机互连形成的，作为统一的计算资源一起进行数据处理，整个机群外部可以看成一台机群进行处理。NUMA 系统通过共享存储器的方式实现多处理机并行工作，其最主要的特点就是系统内某个处理器对其内部的存储器实现了不同速度的读/写，使得对存储字存取时间是随存储器字位置而变化。

2．并行计算机的发展

20 世纪 40 年代开始的现代计算机发展历程可以分为两个明显的发展时代：串行计算时代好并行计算时代。每个计算时代都从体系结构发展开始，接着是系统软件（特别是编译器与操作系统）、应用软件，最后随着问题求解环境的发展而达到顶峰。创建和使用并行计算机主要是因为并行计算机是解决单处理器速度瓶颈的最好方法之一。

20 世纪 60 年代初期，并行计算机多是规模不大的共享存储多处理器系统，即大型主机（Mainframe）。IBM360 是这一时期的典型代表。到了 60 年代末期，同一个处理器开始设置

电子工业出版社

多个功能相同的功能单元，流水线技术也出现了。与单纯提高时钟频率相比，这些并行特性在处理器内部的应用大大提高了并行计算机系统的性能。

当一台 16 个 CPU 系统终于在 1975 年面世时，整个计算机界已经发生了巨大变化。首先是存储系统概念的革新，提出虚拟存储和缓存的思想，其次是半导体存储器开始代替磁芯存储器。1976 年，CRAY-1 问世以后，向量计算机（面向向量型并行计算，以流水线结构为主的并行处理计算机）从此牢牢地控制着整个高性能计算机市场 15 年。CRAY-1 对使用的逻辑电路进行了精心的设计，采用精简指令集，还引入了向量寄存器，以完成向量运算。

由于微处理器的发展，卡内基·梅隆大学率先研制成功一台由 16 个 PDP11/40 处理机通过交叉开关与 16 个共享存储器模块相连接而成的共享存储多处理器系统 C.mmp。从 80 年代开始，微处理器技术一直在高速前进。稍后出现了非常适合 SMP 方式的总线协议，伯克利加州大学则对总线协议进行了扩展，提出了 Cache 一致性问题的处理方案。从此，C.mmp 开创出的共享存储多处理器之路越走越宽。

同一时期，基于消息传递机制的并行计算机开始不断涌现。80 年代中期，加州理工成功地将 64 个 i8086/i8087 处理器通过超立方体互连结构连接起来。此后先后出现了 Intel iPSC 系列、INMOS Transputer 系列、Intel Paragon 和 IBM SP 的前身 Vulcan 等基于消息传递机制的并行计算机。

80 年代末到 90 年代初，共享存储器方式的大规模并行计算机获得了新的发展。IBM 将大量早期 RISC 微处理器通过蝶形互连网络连接起来。人们开始考虑如何才能在实现共享存储器缓存一致的同时，使系统具有一定的可扩展性。90 年代初期，斯坦福大学提出了 DASH 计划，通过维护一个保存有每个缓存块位置信息的目录结构来实现分布式共享存储器的缓存一致性。后来，IEEE 在此基础上提出了缓存一致性协议的标准。

20 世纪 90 年代以来，主要的几种体系结构开始走向融合。随着微处理器、网络设备的发展，以及 MPI/PVM 等并行编程标准的发布，机群架构的并行计算机出现了。IBM SP2 系列机群系统是典型代表，各节点采用的都是标准的计算机，它们之间通过高速网络连接。这种分布存储的并行计算机系统称为机群。并行计算机就进入了一个新的时代，并行计算的应用达到了前所未有的广度和深度。

8.6.2 分布式计算

1. 分布式计算定义

从广义上来看，分布式计算是一门计算机科学，研究如何把一个需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。

从狭义上理解，分布式计算是一种新的计算方式，就是在两个或多个软件互相共享信息，这些软件既可以在同一台计算机上运行，也可以在通过网络连接的多台计算机上运行。

实际上，网格计算是分布式计算的一种，是由一群松散耦合的计算机组成的一个超级虚拟计算机，常用来执行一些大型任务。显然，这种方式具有很强的数据处理能力，网格计算的实质就是组合与共享资源并确保系统安全。

目前常见的分布式计算项目通常使用世界各地上千万志愿者计算机的闲置计算能力，通过互联网进行数据传输。如分析计算蛋白质的内部结构和相关药物的 Folding@home 项目需

電子工業出版社

要惊人的计算量，由一台计算机来完成是不可能的，这时就需要分布式计算。

就目前来看，全球的各种分布式计算已有约百种，这些计算大多互无联系、独立管理、独立使用自己的一套软件。这种分布式计算互相割据的格局很不利于发展的需要。为了改变这种杂乱无章的割据，加州大学伯克利分校提出了建立伯克利开放式网络计算平台（Berkeley Open Infrastructure for Network Computing，BOINC）的想法，把许多不同的分布式计算项目联系起来统一管理，并对计算机资源进行统一分配。对统计评分系统进行统一管理（无论你在为哪个项目工作，只要奉献 CPU 时间长，就积分高）。有了这样的统一管理，就给各个学研究机构提供了便利。目前，多个项目已经成功运行于 BOINC 平台之上。

分布式计算比起其他算法具有以下几个优点：稀有资源可以共享；通过分布式计算可以在多台计算机上平衡计算负载；可以把程序放在最适合运行它的计算机上。其中，共享稀有资源和平衡负载是计算机分布式计算的核心思想之一。

2．分布式计算机系统

分布式计算机系统是指由多台分散的计算机，经互连网络的连接完成分布式计算，系统的处理和控制功能分布在各计算机上。分布式计算机系统的主要特点是：无主从区分，计算机之间交换信息，资源共享，相互协作完成一个共同任务。

① 通信结构：分布式计算机系统的通信结构是指支持各计算机联网，以提供分布式应用的软件。在分布式系统中，尽管每台计算机都有自己独立的操作系统，并且这些计算机和操作系统的种类可以是不同的，但它们都应该支持同样的通信结构。

② 网络操作系统：由网络操作系统提供网络服务功能，分布式系统的硬件环境是计算机网络，系统中的个人计算机可以是单用户工作站或服务器，因此需要由网络操作系统进行管理并提供网络服务功能。

③ 分布式操作系统（透明性）：在分布式系统中有一个公共的分布式操作系统，各计算机共享该操作系统。分布式操作系统由内核以及提供各种系统功能的模块和进程所组成。系统中的每台计算机必须保存分布式操作系统的内核，以实现对计算机系统的基本控制。分布式操作系统除了需要包括单机操作系统的主要功能，还应该包括分布式进程通信、分布式文件系统、分布式进程迁移、分布式进程同步和分布式进程死锁等功能。

① 分布式进程通信：由分布式操作系统所提供的一些通信原语来实现。但由于分布式系统中没有共享内存，这些原语需要按照通信协议的约定和规则来实现。

② 分布式文件系统：允许通过网络来互连，使不同机器上的用户共享文件的系统。它能让运行它的所有主机共享，并可以管理操作系统内核和文件系统之间的通信。

③ 分布式进程迁移：指由进程原来运行的机器（称为原机器）向目标机器（准备迁往的机器）传送足够数量的有关进程状态的信息，使进程能在另一机器上运行

④ 分布式进程同步：在分布式系统中，各处理机没有共享内存和统一的时钟，因此分布式进程同步必须对不同处理机中所发生的事件进行排序，还应该配有性能较好的分布式同步算法，以保证为实现进程同步所付出的开销较小。

⑤ 分布式进程死锁：在分布式系统中，也可能因进程竞争资源而引起死锁，对单处理机系统中讨论过的死锁对策只要稍加修改，就可用于多处理机系统。

3．分布式计算与并行计算的区别及联系

从解决对象上看，分布式计算与并行计算都是运用并行来获得更高性能，大任务化为小

電子工業出版社

任务。它们的区别在于：如果处理单元共享内存，就称为并行计算，反之就是分布式计算。

也有人认为分布式计算是并行计算的一种特例，但是分布式计算的任务包互相之间有独立性，上一个任务包的结果未返回或者是结果处理错误，对下一个任务包的处理几乎没有什么影响。因此，分布式计算的实时性要求不高，而且允许存在计算错误。（因为每个计算任务给好几个参与者计算，上传结果到服务器后要比较结果，然后对结果差异大的进行验证。）

而并行计算处理的任务包之间有很大的联系，并且每个任务块都是必要的，没有浪费的分割，每个任务包都要处理，计算结果相互影响。这就要求每个计算结果要绝对正确，而且在时间上要尽量做到同步。分布式的很多任务块可以根本就不处理，有大量的无用数据块，所以说分布式计算的速度尽管很快，但是真正的“效率”是不高的。

分布式计算与并行计算具体区别和联系如表 8-5 所示。

表 8-5　并行计算与分布式计算对比

对　比		并行计算	分布式计算
相同点		都运用并行来获得更高性能计算，把大任务划分为若干小任务 都属于高性能计算（High Performance Computing，HPC）的范畴 主要目的都在于对大数据的分析与处理	
不同点	时效性	强调	不强调
	独立性	弱，小任务计算结果决定最终计算结果	强，小任务计算结果一般不影响最终计算结果
	任务包之间关系	关系密切	相互独立
	每个节点任务	必要，并且时间同步	不必要，时间没有限制
	节点通信	必须	不必须
	应用的场合	海量数据处理	模式类穷举法

8.6.3　云计算、集群计算及网格计算

1．云计算

云计算（Cloud Computing）是继 20 世纪 80 年代大型计算机到 C/S 模式的大转变之后的又一种巨变，是分布式计算、并行计算、效用计算（Utility Computing）、网络存储、虚拟化、负载均衡、热备份冗余等传统计算机和网络技术发展融合的产物。

美国国家标准与技术研究院（NIST）对云计算的定义是：云计算是一种按使用量付费的模式，提供可用的、便捷的、按需的网络访问，进入可配置的计算资源共享池（资源包括网络，服务器，存储，应用软件，服务），这些资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互。云计算意味着计算能力也可以作为一种商品进行流通，就像水电一样，取用方便，费用低廉，只不过它是通过互联网进行传输的。

云计算有如下特点。

① 超大规模：Google 云计算已经拥有 100 多万台服务器，Amazon、IBM、微软、Yahoo 等的“云”均拥有几十万台服务器。企业私有云一般拥有数百上千台服务器。

② 虚拟化：云计算支持用户在任意位置、使用各种终端获取应用服务。所请求的资源来自“云”，而不是固定的有形的实体。

③ 高可靠性：“云”使用了数据多副本容错、计算节点同构可互换等措施来保障服务的高可靠性，使用云计算比使用本地计算机可靠。

④ 通用性：云计算不针对特定的应用，在“云”的支撑下可以构造出千变万化的应用，同一个“云”可以同时支撑不同的应用运行。

電子工業出版社

⑤ 可扩展性：“云”的规模可以动态伸缩，满足应用和用户规模增长的需要。

⑥ 按需服务：“云”是一个庞大的资源池，按需购买，像水电那样计费。

⑦ 低成本：由于“云”的特殊容错措施可以采用低成本的节点来构成云，“云”的自动化集中式管理使大量企业不需负担日益高昂的数据中心管理成本，“云”的通用性使资源的利用率较之传统系统大幅提升，因此用户可以充分享受“云”的低成本优势。

⑧ 潜在的危险性：云计算服务除了提供计算服务，还提供存储服务。但是云计算服务当前垄断在私人机构（企业）手中，仅仅能够提供商业信用。对于政府机构、商业机构（特别像银行这样持有敏感数据的商业机构）来说，选择云计算服务应保持足够的警惕。另一方面，云计算中的数据对于数据所有者以外的其他用户云计算用户是保密的，但是对于提供云计算的商业机构而言毫无秘密可言。

2．集群计算

在计算机中，集群是使用多个计算机，如典型的个人计算机或 UNIX 工作站，多个存储设备和记忆冗余的互连线路来组成一个对用户来说单一的、高可用的系统。集群计算一个常用用途就是在一个高流量的网站中实现负载均衡，一个网页请求被送到“管理者”服务器，然后此服务器决定此请求由几个相同 Web 服务器中的哪一个进行处理，这样可提升通信量和处理速度。

计算机集群将一组松散集成的计算机软件或硬件连接起来高度紧密地协作完成计算工作。在某种意义上，他们可以被看成一台计算机。集群系统中的单个计算机通常称为节点，通常通过局域网连接。一般情况下集群计算机比单个计算机，如工作站或超级计算机性价比要高得多。

根据组成集群系统的计算机之间体系结构是否相同，集群可分为同构与异构两种。按功能和结构，集群计算机可以分为高可用性集群（High-availability（HA）clusters）、负载均衡集群（Loadbalancing clusters）、高性能计算集群（High-performance（HPC）clusters）、网格计算（Grid computing）。

高可用性集群一般是指当集群中有某个节点失效的情况下，其上的任务会自动转移到其他正常的节点上；还指可以将集群中的某节点进行离线维护再上线，该过程并不影响整个集群的运行。

负载均衡集群运行时，通过一个或者多个前端负载均衡器，将工作负载分发到后端的一组服务器上，从而达到整个系统的高性能和高可用性。这样的计算机集群有时也被称为服务器群（Server Farm）。一般高可用性集群和负载均衡集群会使用类似技术，或同时具有高可用性与负载均衡的特点。

高性能计算集群采用将计算任务分配到集群的不同计算节点儿提高计算能力，因而主要应用在科学计算领域。比较流行的 HPC 采用 Linux 操作系统和其他一些免费软件来完成并行运算。这一集群配置通常被称为 Beowulf 集群，运行特定的程序以发挥 HPC 的并行能力。HPC 集群特别适合在计算中各计算节点之间发生大量数据通信的计算作业，如一个节点的中间结果或影响到其他节点计算结果的情况。

3．集群计算与网格计算

网格计算与集群计算的主要差别如下。

① 网格是连接一组相关并不信任的计算机，它的运作更像一个计算公共设施而不是一

電子工業出版社

个独立的计算机。网格通常比集群支持更多不同类型的计算机集合。

② 网格计算本质上是动态的，集群计算包含的处理器和资源的数量通常是静态的。在网格上，资源可以动态出现，资源可以根据需要添加到网格中或从网格中删除。

③ 网格计算是在本地网、城域网或广域网上进行分布的，而集群计算物理上都包含在一个位置的相同地方，通常只是局域网互连，集群互连技术可以产生非常低的网络延时。物理临近和网络延时限制了集群计算地域分布的能力，而网格计算由于动态特性，可以提供很好的高可扩展性。

④ 集群计算仅仅通过增加服务器满足增长的需求。然而一味地想通过扩大规模来提高集群计算机的性能，它的性价比会相应下降，这意味着我们不可能无限制地扩大集群的规模。而网格虚拟出空前的超级计算机，不受规模的限制，成为下一代 Internet 的发展方向。

⑤ 集群计算和网格计算是相互补充的。很多网格计算都在自己管理的资源中采用了集群计算。实际上，网格用户可能并不清楚他的工作负载是在一个远程的集群上执行的。

尽管网格计算与集群计算之间存在很多区别，但是它们构成了一个非常重要的关系，集群计算在网格计算中总有一席之地。随着网络功能和带宽的发展，以前采用集群计算很难解决的问题现在可以使用网格计算技术解决。理解网格计算固有的可扩展性和集群计算提供的紧耦合互连机制所带来的性能优势之间的平衡是非常重要的。

4．云计算与并行计算

云计算是在并行计算之后产生的概念，由并行计算发展而来，两者之间的区别如下。

① 并行计算、网格计算只用于特定的科学领域、专业的用户，传统并行计算机的使用是一个相当专业的工作，需要使用者有较高的专业素质，多数是命令行的操作。而云计算面向的对象是普通大众，用户可以按需购买所需资源。

② 并行计算追求的是高性能，因此并行计算时代的高性能机群是一个“快速消费品”。而云计算并不去追求使用昂贵的服务器，云中心的计算力和存储力可随着需要逐步增加，云计算的基础架构支持这一动态增加的方式。

③ 并行计算是由单个用户完成的，分布式计算是由多个用户合作完成的，云计算是没有用户参与，而是交给网络另一端的服务器完成。

5．云计算与集群计算

云计算与集群计算的区别在于：集群虽然把多台机器连接了起来，但其某项具体任务执行的时候还是会被转发到某台服务器上；而云计算的任务可以被分割成多个进程在多台服务器上并行计算，然后得到结果，大数据量的操作性能非常好。云可以使用廉价的服务器，可以管理大数据量与大集群，关键技术在于能够对云内的基础设施进行动态按需分配与管理。

電子工業出版社

参考文献

[1] 吴宁．80x86/Pentium 微型计算机原理及应用．北京：电子工业出版社，2000.

[2] 周明德，蒋本珊．微机原理与接口技术（第二版）．北京：人民邮电出版社，2007.

[3] 杨素行等．微型计算机系统原理及应用（第 3 版）．北京：清华大学出版社，2009.

[4] 邹逢兴，陈立刚．计算机硬件技术基础（第二版）．北京：高等教育出版社，2005.

[5] 朱永华等．32 位微型计算机与接口技术．北京：清华大学出版社，2008.

[6] Kip R. Irvine．Intel 汇编语言程序设计（第五版）．温玉杰等译．北京：电子工业出版社，2007.

[7] Intel Corporation．Intel Architecture Software Developer's Manual．Volume 2: Instruction Set Reference. 1999.

[8] Intel Corporation．Intel 64 and IA -32 Architecture Software Developer's Manual．Volume 2A: Instruction Set Reference，2011.

[9] Intel Corporation．Intel 64 and IA -32 Architecture Software Developer's Manual．Volume 2B: Instruction Set Reference，2011.

[10] 钱晓捷．微机原理与接口技术：基于 IA-32 处理器和 32 位汇编语言（第 4 版）．北京：机械工业出版社，2008.

[11] 谢瑞和等．微机原理与接口技术（第二版）．北京：高等教育出版社，2007.

[12] Kunle Olukotun 等．片上多处理器体系结构．汪东升等译．北京：机械工业出版社，2009.

[13] Barry B.Brey．Intel 微处理器结构、编程与接口（第六版）．金惠华等译．北京：电子工业出版社，2004.

[14] Arnold S.Berger．计算机硬件及组成原理（英文版）．北京：机械工业出版社，2006.

[15] 艾德才．微机原理与接口技术（第二版）．北京：高等教育出版社，2009.

[16] 余春暄．80x86/Pentium 微机原理及接口技术．北京：机械工业出版社，2007.

[17] 彭楚武．微机原理与接口技术．长沙：湖南大学出版社，2004.

[18] 许兴存，曾琪琳．微型计算机接口技术．北京：电子工业出版社，2003.

電子工業出版社

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888
传　　真：（010）88254397
E-mail：　dbqq@phei.com.cn
通信地址：北京市万寿路 173 信箱
　　电子工业出版社总编办公室
邮　　编：100036

電子工業出版
PUBLISHING HOUSE OF ELECTRONICS I